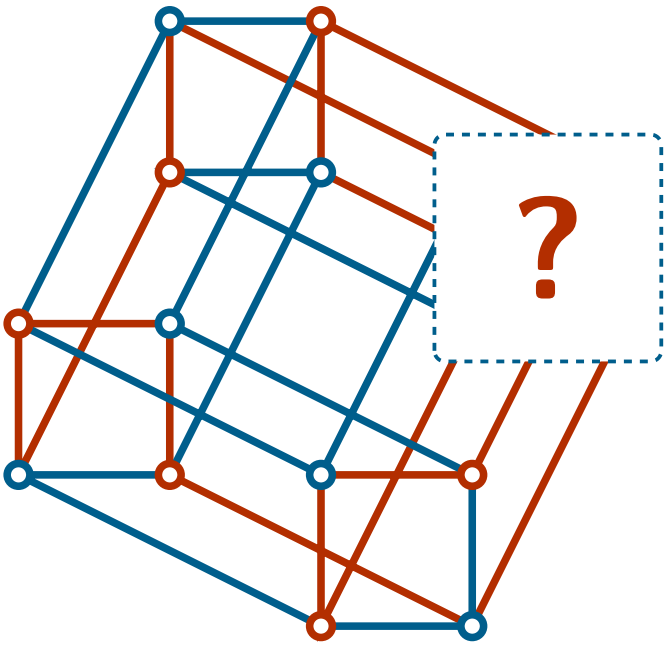
Mikhail Lavrov

Start Doing Graph Theory



stable version (Spring 2026)

available online at https://vertex.degree/

Contents

Preface 8
Reading carefully . . . 8
Definitions . . . 9
Organization . . . 10
Teaching from this book . . . 11
License . . . 12
Do you have any feedback? . . . 12

Acknowledgements 13

The Beginning 14

1 Modeling problems with graphs 15
1.1 Coloring Switzerland . . . 15
1.2 The seven dwarfs . . . 18
1.3 The Towers of Hanoi . . . 20
1.4 A tiling puzzle . . . 23
1.5 Taking photos efficiently . . . 24
1.6 Match the cars and drivers . . . 26
1.7 Practice problems . . . 28

2 Isomorphisms and subgraphs 30
2.1 Circulant graphs . . . 30
2.2 Are these the same? . . . 32
2.3 The isomorphism game . . . 36
2.4 Graph invariants . . . 39
2.5 Subgraphs and some common small graphs . . . 42
2.6 Practice problems . . . 44

3 Walks, paths, and cycles 46
3.1 Walks and paths . . . 46
3.2 Connected graphs . . . 48
3.3 Equivalence relations . . . 51
3.4 Closed walks and cycles . . . 53
3.5 Lengths of walks . . . 55
3.6 Practice problems . . . 58

4 The degree of a vertex 60
4.1 The hypercube graphs . . . 60
4.2 The handshake lemma . . . 61

4.3 Degrees and connectedness . . . 65
4.4 Degrees and cycles . . . 66
4.5 Average degree . . . 68
4.6 Practice problems . . . 70

Vertex Degrees **72**

5 Regular graphs **73**
5.1 Degree sequences . . . 73
5.2 Regular graphs . . . 75
5.3 How many regular graphs are there? . . . 78
5.4 The Petersen graph . . . 81
5.5 The degree/diameter problem . . . 83
5.6 Practice problems . . . 86

6 Graphic sequences **88**
6.1 An unusual party . . . 88
6.2 A graphic sequence algorithm . . . 91
6.3 Two examples . . . 92
6.4 The Havel–Hakimi theorem . . . 94
6.5 More on degree sequences . . . 97
6.6 Practice problems . . . 98

7 Multigraphs and digraphs **101**
7.1 Multigraphs . . . 102
7.2 Degrees in multigraphs . . . 104
7.3 Directed graphs . . . 106
7.4 Directed walks, paths, and cycles . . . 109
7.5 Practice problems . . . 111

8 Euler tours **113**
8.1 Don't lift your pencil! . . . 113
8.2 First steps . . . 115
8.3 Cycle decompositions . . . 117
8.4 Gluing cycles together . . . 119
8.5 Variations on Euler tours . . . 121
8.6 Practice problems . . . 124

Trees **126**

9 Trees and spanning trees **127**
9.1 Spanning trees . . . 127
9.2 Bridges . . . 130
9.3 Properties of trees . . . 131
9.4 Minimum-cost spanning trees . . . 132
9.5 Practice problems . . . 136

10 Properties of trees **138**
10.1 A square garden . . . 138
10.2 Counting edges in trees . . . 140
10.3 From trees to forests . . . 142
10.4 Leaves in trees . . . 143
10.5 Induction on trees . . . 146
10.6 Practice problems . . . 149

11 Cayley's formula **151**
11.1 How to count graphs . . . 151
11.2 Trees and deletion sequences . . . 154
11.3 Prüfer codes . . . 157
11.4 Working with Prüfer codes . . . 160
11.5 Unlabeled trees . . . 162
11.6 Practice problems . . . 163

12 Directed acyclic graphs **165**
12.1 Directed acyclic graphs . . . 165
12.2 Topological orders . . . 168
12.3 Strongly connected . . . 169
12.4 Condensations . . . 171
12.5 Practice problems . . . 174

Matchings **177**

13 Bipartite matching **178**
13.1 Two chess puzzles . . . 178
13.2 Bipartite graphs . . . 180
13.3 Odd cycles . . . 183
13.4 Matching problems . . . 185
13.5 Maximum and maximal . . . 187
13.6 Vertex covers . . . 188
13.7 Practice problems . . . 190

14 Kőnig's theorem **193**
14.1 Kőnig's theorem . . . 193
14.2 Augmenting paths . . . 194
14.3 The augmenting path algorithm . . . 197
14.4 Analyzing the algorithm . . . 199
14.5 Using the algorithm . . . 202
14.6 Practice problems . . . 204

15 Hall's theorem **206**
15.1 Hall's theorem . . . 206
15.2 Regular bipartite graphs . . . 208
15.3 A three-card magic trick . . . 210
15.4 Generalized tic-tac-toe . . . 213
15.5 Incomparable sets . . . 215

15.6 Practice problems . . . 218

16 Matchings in general graphs **220**
16.1 Tutte sets . . . 220
16.2 Saturated graphs . . . 223
16.3 Proof of Tutte's theorem . . . 224
16.4 1-factorizations . . . 226
16.5 Increasing walks . . . 230
16.6 Practice problems . . . 231

Hard Problems **234**

17 Hamilton cycles **235**
17.1 Knight's tours . . . 235
17.2 Two examples . . . 238
17.3 Tough graphs . . . 241
17.4 Sufficient conditions . . . 243
17.5 Practice problems . . . 246

18 Cliques and independent sets **249**
18.1 Two problems about queens . . . 249
18.2 Redundancies and relationships . . . 252
18.3 Greedy algorithms . . . 254
18.4 An indecisive algorithm . . . 256
18.5 Ramsey numbers . . . 258
18.6 Lower bounds . . . 261
18.7 Practice problems . . . 264

19 Graph coloring **266**
19.1 Graph coloring and Sudoku . . . 266
19.2 Greedy coloring . . . 269
19.3 Two lower bounds . . . 271
19.4 Coloring interval graphs . . . 274
19.5 Mycielski graphs . . . 275
19.6 Practice problems . . . 280

20 Line graphs **283**
20.1 Line graphs . . . 283
20.2 Euler tours and Hamilton cycles . . . 286
20.3 De Bruijn sequences . . . 288
20.4 Edge coloring . . . 291
20.5 Vizing's theorem . . . 294
20.6 Practice problems . . . 296

Planar Graphs 298

21 Planar graphs 299
21.1 Three utilities . . . 299
21.2 Planar graphs . . . 301
21.3 Faces . . . 302
21.4 Euler's formula . . . 305
21.5 Barycentric subdivisions . . . 307
21.6 Technical details . . . 309
21.7 Practice problems . . . 313

22 Planarity testing 316
22.1 Counting edges in planar graphs . . . 316
22.2 Triangulations . . . 319
22.3 Girth and planarity . . . 321
22.4 Subdivisions . . . 322
22.5 Graph minors . . . 324
22.6 Overlap graphs . . . 327
22.7 Practice problems . . . 329

23 Polyhedra 331
23.1 The Platonic solids . . . 331
23.2 Classifying the Platonic solids . . . 333
23.3 Dual graphs . . . 336
23.4 Archimedean solids . . . 339
23.5 Practice problems . . . 341

24 Coloring maps 343
24.1 Coloring maps . . . 343
24.2 The four color theorem . . . 345
24.3 Greedy coloring . . . 347
24.4 Five colors . . . 349
24.5 Hamilton cycles . . . 351
24.6 Coloring empires . . . 352
24.7 Practice problems . . . 355

Connectivity 357

25 Cut vertices 358
25.1 Counting paths . . . 358
25.2 Cut vertices . . . 360
25.3 2-connected graphs . . . 362
25.4 Ear decompositions . . . 364
25.5 Induction on ears . . . 369
25.6 Finding common cycles . . . 371
25.7 Practice problems . . . 372

26 Connectivity **376**
26.1 Vertex and edge cuts . . . 376
26.2 Some examples . . . 379
26.3 Local connectivity . . . 382
26.4 Duality . . . 386
26.5 Practice problems . . . 389

27 Menger's theorem **392**
27.1 Menger's theorem . . . 392
27.2 Kőnig-type graphs . . . 393
27.3 Reducing all other cases . . . 396
27.4 Extensions . . . 399
27.5 Cuts and long cycles . . . 403
27.6 Practice problems . . . 405

28 Maximum flows **407**
28.1 Shipping oranges . . . 407
28.2 Maximum flow problems . . . 409
28.3 Network cuts . . . 411
28.4 The Ford–Fulkerson algorithm . . . 413
28.5 Counting iterations . . . 418
28.6 Consequences . . . 421
28.7 Practice problems . . . 423

Appendix **425**

A A review of proof-writing **426**
A.1 Conditional statements . . . 427
A.2 Quantifiers . . . 428
A.3 Unpacking definitions . . . 431
A.4 Optimization problems . . . 433
A.5 Algorithms . . . 435
A.6 The extremal principle . . . 438
A.7 Practice problems . . . 439

B Proof by induction **442**
B.1 The logic of induction . . . 442
B.2 Induction on graphs . . . 446
B.3 Inductive definitions . . . 449
B.4 Recursive constructions . . . 452
B.5 Practice problems . . . 456

C License **458**

D Bibliography **464**

E Index **470**

Preface

The purpose of this chapter

I have written very many words about graph theory; in the internet age, it is very easy to rack up a frankly embarrassing word count. However, I have never written a textbook before. To address any shortcomings, I am trying to be very deliberate, and tell you what I am doing and why, in the hopes that if I fall short, someone will tell me so that I can improve. At the beginning of every chapter, I have a section telling you what I am trying to accomplish in it.

Similarly, in this chapter, I want to tell you what I'm trying to do in the book as a whole. Most of it is about specific details of this textbook; how I write, why, and how you should and shouldn't read it. I've also included a bit of general advice on reading a math textbook or teaching from one.

Reading carefully

This textbook is an introduction to graph theory, but rather than calling it "Introduction to Graph Theory", I have called it "Start Doing Graph Theory", and not just because the first title was already taken. It is because you should start doing graph theory if you want to learn it. I will try to help you along with this by telling you the story of how to solve a problem, whenever I can. You should cooperate by trying to solve problems, whenever you can. This includes, but is not limited to, the practice problems at the end of every chapter. Moreover, I've included many question-and-answer boxes that look like this:

Question:	What is $2 + 2$?
Answer:	4, probably.

Whenever you see one, you should pause to ask yourself the question and confirming that you understand how to arrive at the answer. I have borrowed the use of such question-and-answer pairs from Mor Harchol-Balter's *Performance Modeling and Design of Computer Systems* [51].

This textbook is a proof-based introduction to graph theory: I want to teach you how to prove theorems about graphs. Many proofs of theorems about graphs are included in this textbook, because you need to see how the thing is done before you do it. (A mathematician should have a good memory; often, the way to solve a problem is by realizing that a trick you've seen before in the proof of an unrelated statement applies to your problem, too.) When I present proofs in this textbook, I try to help you along by explaining how we might arrive at the strategy we take. However, once again, you will not start learning about proofs in graph theory until you start writing proofs in graph theory.

If you're reading the chapter or listening to a lecture, try to anticipate the next step before it happens. When you're done, try solving the practice problems that tell you to prove something. It is insufficient to think about a problem until you're convinced you know how to solve it. Carefully writing down a proof can reveal errors and the gaps in your thinking, and so you should carefully write down your solutions. I have personally discovered counterexamples to my conjectures by trying to write down proofs of them.

You will notice, if you haven't already, that this book uses pronouns more often than most textbooks. Mathematicians already bring something unconventional to the table with the use of "we"; I've made things even worse by using "you" and "I". Why?

- The role that "I" take on in this textbook is the role of the writer. I have not personally discovered all of graph theory. However, in this textbook, I have ultimate power to decide which topics to put where, and so I write in the first person when I write about these decisions.
- The role that "you" take on in this textbook is the role of the reader. I write in the second person when I talk about what you have read, what you're about to read, and what you might read elsewhere. More ambitiously (and I hope you will forgive me if I overstep) I write in the second person when I suggest to you what you should be thinking about as you read.
- This leaves the role that "we" take on together: the role of the mathematician. When we prove a theorem, we do it together, and it should be an active process on your part. You should keep our common goals in mind, and think about how to get from where we are to where we'd like to be. If I present an argument, your job is to be as skeptical as possible: we have only proven a theorem once you are convinced that the proof I wrote down is airtight, or once you've done your own work to fill in any gaps. (This is much harder when you write proofs of your own! In that case, you need to play the game against yourself, and be immediately skeptical of the words you have just written.)

The book would be ten times longer if I presented every proof the way that mathematics is actually done, because it is very rare to stumble on the correct approach from the very beginning: it is much more common to fail nine times and succeed on the tenth try. So my last piece of advice to you in this section is this: don't get discouraged by mistakes.

Definitions

Graph theory is a definition-heavy subject; there are over 200 definitions in this book. I don't think it should be a priority to memorize all the definitions you encounter, and that also seems like a miserable way to engage with graph theory. In many cases, when you go out to do math on your own, you will be able to look up all the definitions you need, as you need them. You just have to remember that a concept exists and what it's for, or else you will not know to look it up.

That being said, internalizing a definition can be very useful; I mean not just memorizing the definition, but having a clear idea of how the parts of that definition fit together and what they're good for. A definition that you need to look up is an obstacle to overcome; a definition that you've internalized is another tool in your hands. You will naturally internalize definitions

you work with frequently, and which ones this will be depends on what you end up working on.

To make the definitions in this book less overwhelming, I have introduced three "tiers" of definition to help you prioritize. The most important definitions are those set aside in their own numbered paragraph, like so:

Definition 3.14. *This is the definition of* ***concept****.*

These are definitions that you will need many times throughout the book, so you had better learn them well.

In the next tier are the terms localized to a single part of the textbook. For example, there are concepts that are important to the study of matchings, which we will frequently use in Chapters 13 through 16 but rarely refer to again. In such cases, I will merely give an in-line definition, writing the new term **in bold**. These terms are less universal, but they are particularly important in their sub-field of graph theory. (At the end of a part of the book, such as in Chapter 8, there seems to be no difference between this tier and the third tier, so I will promote a few definitions to the second tier solely based on importance.)

Finally, the last tier of definitions will mainly be useful to us in a single chapter, with at most a brief mention later. This does not mean that these concepts are not important to anyone; this tier also includes important but specialized concepts from topics I won't cover in more detail. On the other hand, some of these definitions are just meant to make a proof or two easier to read by giving a recurring concept a name; in a few cases, which I will mention as they come up, I even made up the name myself. Anyway, I will *italicize* such a term, so you know that you should pay attention to it but not necessarily commit it to your long-term memory. Also, if I mention a concept before it is defined, or after we've left its "scope", I will *italicize* it and give you a brief definition with a chapter reference. This is very common in Chapter 1, which mentions many topics in passing which I won't really introduce until later in the book.

I may sometimes mention a new piece of terminology without so much as italicizing it. This means that, in this textbook, I will not rely on you to know that concept by name. It does not mean that the concept is unimportant outside this textbook, and in fact I probably mentioned it to let you know that some mathematicians do care about it. You will still be able to find such terms in the index.

Organization

This book is divided into 7 parts and an appendix. The first part (Chapters 1 through 4) are an introduction to the basic concepts of graph theory. In my original vision, the other parts of the book were meant to be independent, so that you could read any of them after the first part and be fine. Within each part of the book, you should definitely go in order.

In reality, this didn't quite work out, but there are only a few fundamental departures from this organizational system, so I can simply list them all here.

- Multigraphs and directed graphs (defined in Chapter 7) are mentioned fairly frequently later on, but for the most part, it is just to explain how what we're doing can be extended to those settings, which is skippable. One exception is Chapter 12, which I feel belongs in the part on trees in a spiritual sense, but is really all about directed graphs.
- In Chapter 20, matching problems (from Chapter 13 and Chapter 14) are used several times. If you're interested in the discussion of Euler tours and de Bruijn sequences, Chapter 7 and Chapter 8 are also a necessary prerequisite.
- Chapter 24 is all about coloring planar graphs, which won't make sense without the prior discussion of graph coloring in Chapter 19.
- Chapter 27 relies heavily on König's theorem from Chapter 14 to prove Menger's theorem, and on line graphs from Chapter 20 for the edge version of Menger's theorem.
- Chapter 28 is presented completely in the language of directed graphs, from Chapter 7.

Except for these cases, I may reference earlier chapters but I don't rely on it. Sometimes I use a topic from a different part of the book as an example, or even dedicate a section of a chapter to exploring the connections between two topics. (In the note at the beginning of the chapter, I will point this out.) I think that you will be missing out on some valuable math if you have to skip them, but it will not get in the way of you understanding the current material.

I say all this to make it clear what your options are, but I still think it's best if you go in order; that's why I put the chapters in the order I did! In particular, I think that the topics in the last part of the textbook (Chapters 25 through 28) is a good part to leave until the end. In those chapters, I have pointed out many connections to previous parts of the textbook, serving as a review of those topics.

What about the appendix? Appendix A is a review of proof-writing, and Appendix B is about proof by induction. Both of these are going to be important throughout the book, but whether you need to read them or not depends on your background. I have written the two chapters of the appendix so that they only rely on topics from the first part of the book; if you're going to read them, you should do so either alongside or right after Chapters 1 through 4.

Teaching from this book

Now, let me address the instructors of introductory graph theory courses; if you are a student or self-studying, feel free to take this opportunity to peek back-stage, as it were.

I have based this textbook on courses I have taught, in various formats, at the University of Illinois Urbana-Champaign, at Kennesaw State University, and at a Canada/USA Mathcamp (a summer program for high school students). Some individual sections of this book have led migratory lifestyles, moving between all three of these places along with me. The first time I wrote most of this down, however, was when I taught graph theory at KSU. As a result, most chapters in this book correspond to individual days of a 75-minute university class. This does not mean that you should plan to cover the entire contents of a chapter in one class! Almost all of these chapters have expanded as I've adapted them to a textbook format.

That being said, you can still probably get pretty far on a one-class-per-chapter basis. What I often do is pick a minimum amount of material I want to be certain I'll get through over

the course of 1–2 classes: I base this on what I'll want my students to know in the future. Everything else in my plans is nice to have, but not necessary. Sometimes that means topics I think are cool; sometimes it means I want to do a few trickier proofs or extended examples at some point, but I don't have to do it with this particular topic.

If I have extra time and I seem to have successfully covered the minimum, then I move on to the bonus topics. If have to slow down, because I get lots of questions, or because the material doesn't seem to be landing, or because I messed up a proof, then hopefully I still get through the minimum I set myself, which lets me move on without rearranging the overall schedule. (If I find that I never get through more than the minimum, though, then maybe I'm being too ambitious. I'd rather teach less graph theory overall than never show a tricky proof, and a regular dose of cool extra topics is important too.)

If you feel that a topic is particularly important, and want to address it in detail, then I generally think it's reasonable to budget two classes for that chapter rather than one. This is especially true if you want to include time for students to solve problems in class, and even more true than that if you want to do any amount of teaching by discovery. I would consider, in that case, using some of the practice problems at the end of the chapter as topics of in-class discussion.

Please get in touch with me if you are teaching from this book and would like solutions to the practice problems. At some point in the future, I might also add hints and solutions to some of the problems to the book itself.

License

This book is licensed under the Creative Commons Attribution-ShareAlike 4.0 (CC BY-SA 4.0) license. You can read it in full at the end of this book or at the URL below:

http://creativecommons.org/licenses/by-sa/4.0/

This means that you're welcome to download, read, and share this book; if you share it, you must give credit to the author (Mikhail Lavrov), and if you make any copies or derivatives of this book, they must be released under the same license.

Do you have any feedback?

If so, please get in touch! I would like to hear from all of my readers, so please write to me at misha.p.l@gmail.com whether you've used it as a course textbook, or merely stumbled on it randomly and decided to read it, or whether you've thought better of it and decided *not* to read it, or anything in between. If you're thinking, "But surely he doesn't mean me," then you're wrong; I do mean you. If you're thinking, "I bet he gets too much email already," then don't worry about it; if too many people write to me, I'll change this paragraph accordingly.

I especially want to know if there are any mistakes in this book, minor or major, so if you find any, please let me know.

Acknowledgements

For the motivation to write this book, I would like to thank all of my graph theory students: at the University of Illinois Urbana-Champaign, at Canada/USA Mathcamp, and at Kennesaw State University. You've asked many good questions, some of which I had no answers to; you've been patient with me when I made mistakes, you've laughed at my jokes, you've been skeptical at all the right times. Some of you made me a bucket hat with the volcano graph on it, and some of you have hijacked my laptop to edit typos into my lecture notes. All of you are the reason why this book is worth writing.

My PhD advisor, Po-Shen Loh, has been an influence on and a source of inspiration for my teaching for many years. In this book, you might recognize some of his favorite problems. I also have to say that my first time teaching several of the topics in this book was as a TA for Po's Discrete Mathematics class at Carnegie Mellon University—and I learned how to teach those topics for the first time by watching Po teach. As a reward for reading these acknowledgements, here is the homework problem of Po's that I've watched the most students struggle with: "Does every graph which contains two edge-disjoint Hamilton paths also contain a Hamilton cycle?" So that Po's students continue to struggle with it, I will not tell you the solution.

I've been influenced by many authors of textbooks, in graph theory and outside it. In Douglas West's *Introduction to Graph Theory* [104], every definition, example, and proof is incredibly thoughtfully chosen, and in some cases, once you see his approach, there is no question of doing anything else. Richard Trudeau's *Introduction to Graph Theory* [97] could not be a more different book despite sharing the same title, and even though I disagree with many of the pedagogical decisions in it, it explains some topics more clearly than any other book I've read. Clive Newstead was a fellow graduate student at Carnegie Mellon University when he started work on his *Infinite Descent into Pure Mathematics* [77], which made the frankly ridiculous goal of writing a book myself seem slightly less ridiculous; Clive was also very helpful in answering some of my questions about writing a textbook. I have already mentioned Mor Harchol-Balter's *Performance Modeling and Design of Computer Systems* [51], but I have to also say that Mor herself contributed a lot to my growth as a teacher and a mathematician.

I am grateful to Ben Dees, Allison Hung, Jennifer Vandenbussche, and Solomon Methvin for finding the first four typos of what I am sure is a much longer, yet-to-be-uncovered list.

My wife, Elizabeth, has been endlessly patient with me throughout the process of writing this book, even when I've stared off into space blankly thinking about graph theory, or asked her inane questions like, "Do you think this diagram looks better if I move this vertex a little to the left?" Elizabeth, you're great. Thank you.

The Beginning

1 Modeling problems with graphs

The purpose of this chapter

Before you begin learning about specific problems in graph theory, I want to convince you that graph theory is useful. Well, of course I would say that—I'm a graph theorist! However, one of the reasons I'm firmly convinced that graph theory is one of the most highly applicable areas of math is that I often find it fruitful to think about the world graph-theoretically even when I'm not trying to solve problems in graph theory.

This comes easier to me, because I've spent many years with all this graph-theoretic language at my fingertips. It will not come easily to you at first, when you don't know any graph theory. So one of them things I want to do here is to show you how I think about turning a problem into a graph. Stop and think about it yourself! Think about it with each new example.

I want to limit myself to examples that are simple enough to fully explain in this chapter. However, I also want to give complete examples that I can fully describe to you. Finally, I want to give you a variety of examples: I want to show you different types of problems in graph theory, and I want to show you very different flavors of applications, as well.

I will ask many questions about the graphs in this chapter, and I will often give formal mathematical names for the answers to those questions. However, I will not answer the questions here, and you should not feel obligated to learn any of the fancy terminology yet: the only terms I am asking you to learn at this point are those set aside in a "**Definition 1.x**" paragraph. If you become curious about the other ideas mentioned here—good! However, you will have to wait until a later chapter.

1.1 Coloring Switzerland

Switzerland (formally known as the Swiss Confederation) is made up of 26 cantons. On a map, you will often see these drawn in different colors, to make the borders between different cantons clearer to see. As a result, it's particularly important to choose the colors for each canton so that adjacent cantons receive different colors.

I have used five different colors to color the map in Figure 1.1. This is a bit unfortunate, because the color palette I've picked out to use throughout this book contains only four colors. Several of the cantons are given a checkerboard pattern instead of a solid color, which makes it somewhat harder to see where the borders lie.

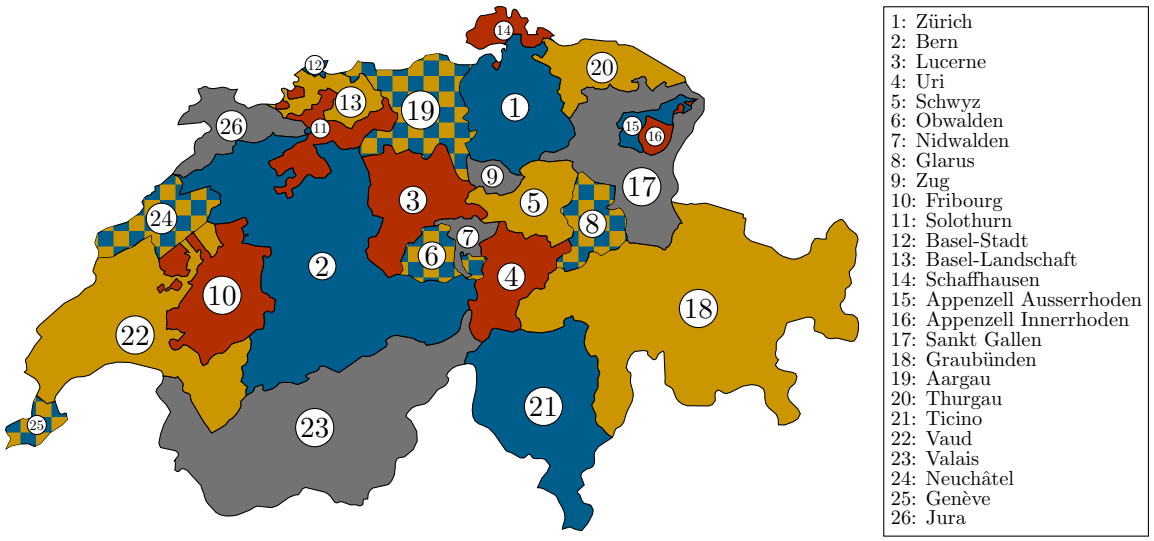


Figure 1.1: A map of Switzerland (coordinates from [92])

Could we color the map of Switzerland with four colors?[1] And what is the minimum number of colors needed? The answer to that question is called the chromatic number of the graph that describes the problem—but to say that properly, we first have to say what a graph is.

This is not the same "graph" that you would encounter in high school algebra! Blame the Greeks: in Greek, the root "graph" just means "picture", and there are quite a few things in mathematics that we want to draw pictures of, so there is some overlap in terminology as well.

Unfortunately, the word is particularly awkward in the context of graph theory, because here the graphs are specifically not pictures! Instead, a graph is going to be exactly the mathematical object that we need to ask questions like, "How many colors are necessary to color the cantons of Switzerland?" Before giving the definition, it's worth thinking about the things we do and don't need to know:

- We don't need to know that the canton of Thurgau is in the north part of eastern Switzerland, nor that it's shaped roughly like a shark (in my opinion).
- We also don't need to know that it's called "Thurgau", as opposed to "Thurgovia" (its anglicized name) or "20" (the numerical label it is given in Figure 1.1) or anything else. We will need to refer to the cantons individually somehow, or else it would be very hard to talk about which cantons get which colors. However, it doesn't matter at all which names we give them.
- We do need to know that Thurgau (or 20, or whatever we choose to call it) borders Schaffhausen (14), Sankt Gallen (17), and Zürich (1): if we want to color the map so that adjacent cantons get different colors, then these are the colors we'll need to distinguish.

For example, consider the diagrams in Figure 1.2, in which the cantons have been replaced by short numerical labels or even just plain dots, and their placement only loosely reflects their geographic location. This is just as good for our purposes; maybe even better, because it does not include any distracting details!

[1] If you already know a bit of the relevant theory, and think that you've heard of a powerful result that solves this problem: Switzerland has a bit of a surprise for you! The full solution takes a bit more work.

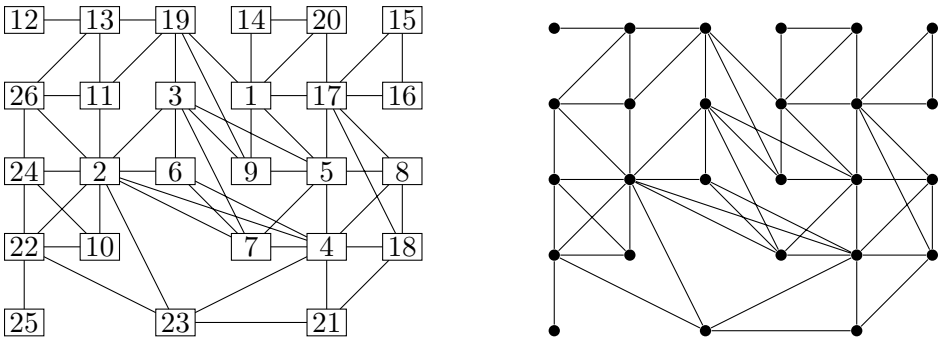


Figure 1.2: Two schematic representations of the Switzerland graph

Even having a diagram at all is a concession to our human needs. If we wanted to try to use a computer to solve the problem, we might simply enter the following (which, under the hood, is how the two diagrams in Figure 1.2 are generated):

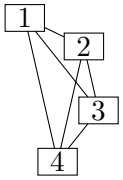
```
(1) -- (5) (1) -- (9) (1) -- (14) (1) -- (17) (1) -- (19) (1) -- (20)
(2) -- (3) (2) -- (4) (2) -- (6) (2) -- (7) (2) -- (10) (2) -- (11)
(2) -- (22) (2) -- (23) (2) -- (24) (2) -- (26) (3) -- (5) (3) -- (6)
(3) -- (7) (3) -- (9) (3) -- (19) (4) -- (5) (4) -- (6) (4) -- (7)
(4) -- (8) (4) -- (18) (4) -- (21) (4) -- (23) (5) -- (7) (5) -- (8)
(5) -- (9) (5) -- (17) (6) -- (7) (8) -- (17) (8) -- (18) (9) -- (19)
(10) -- (22) (10) -- (24) (11) -- (13) (11) -- (19) (11) -- (26)
(12) -- (13) (13) -- (19) (13) -- (26) (14) -- (20) (15) -- (16)
(15) -- (17) (16) -- (17) (17) -- (18) (17) -- (20) (18) -- (21)
(21) -- (23) (22) -- (23) (22) -- (24) (22) -- (25) (24) -- (26);
```

With that in mind, let us finally give the definition of a graph, which will reflect all of these thoughts and only keep what is truly important. This definition can be used in many ways, to capture all kinds of relationships between all kinds of objects; I hope that the examples in this chapter give you an intuitive understanding of how to use it.

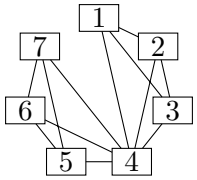
Definition 1.1. *A **graph** G is a pair (V, E) where:*

- *V is a set of arbitrary objects called **vertices**; a single one is called a **vertex**. In the Switzerland graph, V is the set of the 26 cantons. We write $V(G)$ for the set of vertices of a graph G.*
- *E is a set of **edges**: each edge is an unordered pair of vertices, or just a set of two vertices. In the Switzerland graph, E is the set of all pairs of cantons that share a border. We write $E(G)$ for the set of edges of a graph G.*

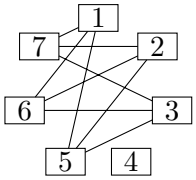
Since an edge is a set of two vertices, it should be written with the notation $\{x, y\}$, but to avoid too many symbols in notation that is used all the time, graph theorists often skip the curly brackets and just write xy. I will do the same in this book, only using set notation when it's necessary to avoid ambiguity. For example, we should not represent an edge in the Switzerland graph as "117", because it's not clear if this means $\{1, 17\}$ or $\{11, 7\}$.



(a) The first card game



(b) The first two games



(c) What's left

Figure 1.3: The seven dwarfs

There is a lot of secondary terminology to describe the relationship between edges and vertices in a graph. Here are a few of the most important terms:

Definition 1.2. *Vertices x and y in a graph G are **adjacent** when G has an edge xy; we also say that x is a **neighbor** of y (and vice versa).*

Definition 1.3. *The **endpoints** of the edge xy are the vertices x and y. An edge is **incident** to its endpoints, and vice versa: x and y are incident to xy, and xy is incident to x and y.*

It's also okay to say that an edge xy goes between x and y, or joins x and y, for example; these are not really technical terms, but just English words used in the ordinary way to convey relationships in a graph. As long as it's clear what you mean, the exact words you use are flexible. Avoid using words like "connect" or "connected" informally, though: they have a technical meaning in graph theory.

In Chapter 19 we will define a *coloring* of a graph G to be a function from $V(G)$ to some set C whose elements we will call "colors". An actual non-mathematical coloring of a map, such as in Figure 1.1, can be described by such a function: for example, $f(20) =$ blue might describe coloring Thurgau blue. In this model of coloring, we will see how to prove lower and upper bounds on the number of colors needed.

1.2 The seven dwarfs

One day, the seven dwarfs of fairytale[2] wanted to play cards. Unfortunately, they only have one deck of cards. Their favorite card game, Hearts, is a four-player game, so only four dwarfs can play at a time. If the dwarfs switch in and out of the game, how many rounds will they need to play so that every dwarf has gone up against every other dwarf at least once?

It seems reasonable to start with two games that don't overlap very much: for example, a game with dwarfs 1–4 followed by a game with dwarfs 4–7. After that, though, we might need some help keeping track of which dwarf has already played which other dwarf. Since this is a property of different pairs of dwarfs, it is natural to model it with a graph whose vertices are the dwarfs. Figure 1.3a shows the pairs of dwarfs that face each other in the first game; Figure 1.3b shows the status after both games; finally, Figure 1.3c shows the pairs of dwarfs that have yet to face each other.

[2]To avoid any potential entanglements with Disney, I will use the traditional names for the seven dwarfs: Dwarf 1 through Dwarf 7.

I admit that I chose this example with an ulterior motive: to illustrate a few common operations on graphs. Graphs are built out of sets, and the fundamental operations of set theory—union, intersection, and complement—all show up graph theory, as well.

The graph in Figure 1.3b is assembled out of two graphs: the graph in Figure 1.3a which tracks only the first card game, and a similar graph (not pictured) which tracks only the second card game. This is the operation we'll define to be the union of two graphs.

Definition 1.4. *The* ***union*** $G \cup H$ *of two graphs* G *and* H *is the graph with all vertices and edges appearing in at least one of the two graphs:* $V(G \cup H) = V(G) \cup V(H)$ *and* $E(G \cup H) = E(G) \cup E(H)$.

The union $G \cup H$ *is called a* ***disjoint union*** *if* G *and* H *share no vertices or edges.*

Question:	Is the union in Figure 1.3b a disjoint union?
Answer:	No: even though the graphs of the two card games share no edges, vertex 4 appears in both graphs.

We could also define the graph intersection $G \cap H$ to be the graph that only keeps the vertices and edges common between G and H, but that is a much less common operation.

Finally, we get to the complement. This is an awkward operation to define in set theory. If $\overline{S}$ is the set of all objects not appearing in S, then we can probably all agree that a complement like $\overline{\{1, 2\}}$ contains the element 3, but does it contain the element π? What about the color purple? So in serious set theory, it's much more common to use the relative complement, or set difference, $A - B$: the set of all objects appearing in A, but not B. We will see plenty of set differences in this book too, but fortunately, there is a natural notion of complement when it comes to graphs.

Question:	In what sense is the graph in Figure 1.3c a complement of the graph in Figure 1.3b?
Answer:	The graphs have the same 7 vertices, but the graph in Figure 1.3c has exactly the edges that Figure 1.3b does not have.

This is the definition we make in general:

Definition 1.5. *The* ***complement*** $\overline{G}$ *of a graph* G *is the graph with* $V(\overline{G}) = V(G)$ *such that two vertices* x *and* y *are adjacent in* $\overline{G}$ *exactly when they are not adjacent in* G.

It would be unkind of us to use the dwarfs as an example of unions and complements without addressing their concerns about how to schedule their card games. Graphs can help with that as well. We can get a preliminary answer just by counting edges.

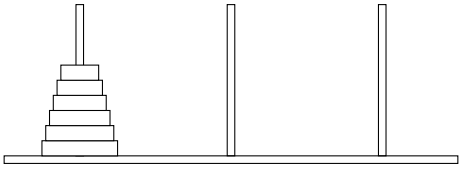


Figure 1.4: The Towers of Hanoi puzzle

Question:	How many pairs of dwarfs are there, and how many face each other in each game? What does this tell us?
Answer:	Altogether, there are $\binom{7}{2} = \frac{7 \cdot 6}{2} = 21$ pairs[3] of dwarfs: we choose two dwarfs in $7 \cdot 6$ ways, and divide by 2 because order doesn't matter. In each card game, $\binom{4}{2} = 6$ dwarfs face each other, so at least 21/6 games are necessary, which rounds up to 4.

Unfortunately, if the dwarfs play two games as in Figure 1.3, they can't finish by playing two more. A single card game can only take care of 4 of the 12 remaining edges in Figure 1.3c, so at leas three more card games are needed. (See if you can find 3 card games that will do it!) We can't avoid arriving at the graph in Figure 1.3c entirely, either! It occurs whenever there are two card games that only have one dwarf in common. But if this never happens, then every dwarf needs at least three games to face all 6 other dwarfs, and this leads to an even longer solution. Ultimately, at least five card games are necessary, however we schedule them.

1.3 The Towers of Hanoi

The Towers of Hanoi puzzle was invented in 1883 by Édouard Lucas [6]; it is now so popular its origins have been nearly forgotten. In this puzzle, you have three pegs, and some number of disks of different sizes stacked on the pegs. Initially, all the disks are placed on one peg, sorted by size (with the smallest disk on top), as shown in Figure 1.4.

You are allowed to move the disks in a limited way: in a single move, you can lift the top disk on a peg, and put it down on another peg, provided you do not place a larger disk on top of a smaller one. Placing a larger disk on a smaller one is always forbidden. The goal of the puzzle is to move all the disks from one peg to another.

How can we model this puzzle as a graph? This is a much trickier question than in the previous section, and the answer may be counter-intuitive the first time you see something like it. When I've asked the question to clever students new to graph theory, they've been tempted to start with either the disks or the pegs as the vertices, which turns out not to lead us anywhere fruitful.

To arrive at a useful answer, I suggest thinking as follows: how can we define a graph that will know everything about the rules of this puzzle, so that when we try to use graph theory to solve it, we will not need to know anything? Our graph does not need to be a small, efficient

[3] If you haven't seen binomial coefficients $\binom{n}{k}$ yet in your mathematical career, the only one you'll need to know for 99% of graph theory is $\binom{n}{2} = \frac{n(n-1)}{2}$: this counts the number of unordered pairs of n objects, or the number of possible edges an n-vertex graph could have.

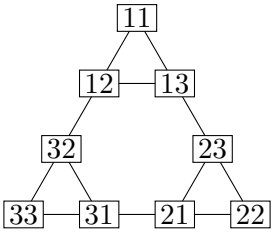


Figure 1.5: The graph representing a 2-disk Towers of Hanoi puzzle

representation of the rules, either. On the contrary: if the solution to the puzzle (which may be a very long sequence of moves) is to be found anywhere in the graph, then the graph will have to be pretty big.

The strategy we take to model the Towers of Hanoi puzzle as a graph G is to do the following:

- Let the vertex set $V(G)$ be the set of all possible states of the puzzle. For example, the picture in Figure 1.4 is just one of the vertices. If we take the smallest disk and move it to the middle peg, the result is a different state of the puzzle, represented by another vertex. Our final goal is to have all the disks stacked up on a different peg, which is yet another vertex.
- Let the edge set $E(G)$ be the set of all possible pairs xy where it's possible to turn state x into state y by making a single move. Or, phrased more concisely: let two states $x, y \in V(G)$ be adjacent if a single move can turn x into y.

  (This is a symmetric relationship: if we can move a disk from one peg to another, we can always move it back.)

Question:	In a graph, our edges should be unordered pairs, but the rule "a single move can turn x into y" seems like an ordered relationship. Is this a problem?
Answer:	Not in this case, because if we can move a disk from one peg to another, we can always move it back, so the relationship is symmetric. Puzzles where we cannot always reverse a move we make are more difficult to model.

As you might imagine, this definition of $V(G)$ results in a large graph G indeed. So in Figure 1.5, the graph G is illustrated for a version of the puzzle with only 2 disks: a small one and a large one. The two-digit label on each vertex records the positions of the two disks: the first digit records the position of the bigger disk, and the second digit records the position of the smaller disk. In this way, the two-digit number describes the state of the two-disk Towers of Hanoi puzzle. (In our definition, we said that $V(G)$ is the set of all possible states of the puzzle, but we didn't specify how the states of the puzzle are encoded. That's because it doesn't really matter!)

Question:	In Figure 1.5, which vertices represent our starting state and our final state?
Answer:	The vertices 11, 22, and 33 are the three vertices in which both disks are stacked on the same peg; we want to get from one of these to another.

Question:	What do the edges from vertex 12 to vertices 11, 13, and 32 represent?
Answer:	The smaller disk can be moved from peg 2 to peg 1 or peg 3; these moves give us the edges from 12 to 11 and 13. The bigger disk can only be moved from peg 1 to peg 3, giving the edge from 12 to 32; it cannot be moved to peg 2, because it cannot be placed on top of the smaller disk.

Now that we have the graph, how do we think about solving the puzzle? To answer that question, let's think about what happens in the world of graph theory when we attempt to solve the puzzle by moving disks around. Each time we lift a disk and move it to a different peg, we go from one state of the puzzle to another: from one vertex to another. Suppose we make m moves; then we can imagine recording the entire history of the moves we've made as a sequence

$$(x_0, x_1, x_2, \ldots, x_{m-1}, x_m)$$

in which each term x_i is a vertex in the Towers of Hanoi graph. The moves that we've made are valid moves if it's the case that for all $i = 1, \ldots, m$, the pair $x_{i-1}x_i$ is an edge of the Towers of Hanoi graph.

In graph-theoretic terminology, such a sequence is called a *walk* from x_0 to x_m, or an "$x_0 - x_m$ walk" for short; see Chapter 3 for more details. When we're talking about puzzles like the Towers of Hanoi, that's a very metaphorical walk: you could picture a little figure standing on the diagram in Figure 1.5 and walking along the edges, but that figure exists only in your imagination. It would become a much more literal walk in the Switzerland graph from the previous section! Suppose that you live in Zürich, and you decide to go on a hike through Switzerland until you reach Geneva. Then the sequence of cantons you walk through (updated every time you cross a border between cantons) will be a walk in the Switzerland graph: a $1-25$ walk where 1 is the Canton of Zürich and 25 is the Canton of Geneva.

A solution to the Towers of Hanoi puzzle is a walk with very specific starting and ending vertices: we want to start in a vertex x_0 corresponding to all disks being on a single peg, and we want to end in a vertex x_m corresponding to all disks being on some other single peg. The *length* of this walk, m, is the number of moves that we needed to make, so our second goal might be to minimize this length and find the shortest solution.

The question of finding minimum-length walks in a graph is not only useful for puzzles. If you were to ask your phone for walking directions from Zürich to Geneva, it would consult a much bigger graph: the graph of possible pedestrian locations in Switzerland, at a fairly low resolution, and with additional metadata such as walking times that we're not considering yet. However, in that bigger graph, it would solve essentially the same problem of finding a minimum-length walk!

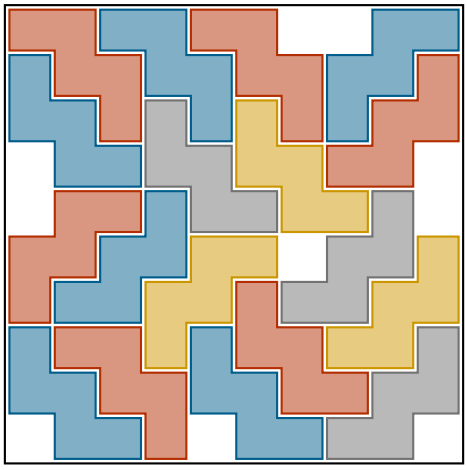


Figure 1.6: A solution to fitting 18 zigzag tiles in a 10×10 grid without overlap

1.4 A tiling puzzle

Here is a different kind of puzzle. (I promise that graph theory is not just good for puzzles! However, puzzles are a very convenient application: they usually have much simpler rules than the real world, so we can describe them cleanly using graph theory, and puzzle creators often at least try to make their puzzles fun.)

Suppose that you have an infinite supply of shaped tiles. How many of them can you place on a 10×10 grid without overlap? One possible optimal solution to this problem is shown in Figure 1.6.

How did I find this solution? I did it by encoding the problem as a graph and using some tools in Mathematica's graph theory library. It's not obvious how to represent this as a graph theory problem, but here is what I ended up doing:

- Let the vertices be all ways to place a single tile on the 10×10 grid.
- Put an edge between two vertices if the tile placements they represent are incompatible: the tiles would overlap.

Question:	How many vertices does this graph have?
Answer:	The middle square of a tile can be in any of the $8^2 = 64$ squares that are not on the edges of the 10×10 grid. Once we pick where the middle square goes, there are 4 ways to orient the tile, for a total of $64 \cdot 4 = 256$ placements.

A solution to the puzzle is a collection of locations where we've placed tiles, so it's a set of vertices in this graph.

Question:	Which sets of vertices are valid solutions to the puzzle?
Answer:	A set of vertices tells us where to place tiles, but it's only a valid solution if none of the tiles we place overlap. Overlapping tiles are represented by edges, so we want a set in which no two vertices are adjacent.

A set of vertices with no edges between them is called an *independent set* in a graph, and we will look at the problem of finding these in Chapter 18.

In fact, there is a second way to think about the problem that's equally good. We could have defined two vertices to be adjacent if the tiles would not overlap: this graph would be the complement of the graph we originally defined. In the complement graph, a conflict-free solution corresponds to a *clique*: a set of vertices in which any two vertices are adjacent. Cliques and independent sets are both important; though the problems they solve are equivalent, as we see here, sometimes one is more natural than the other to think about, and sometimes we study the relationship between the two problems.

Finding the largest independent set in a 256-vertex graph is a lot for a human, but not out of the reach of computer algorithms. It took my laptop 48 seconds to find the largest independent set in the graph: less time than it took me to describe the problem to my computer!

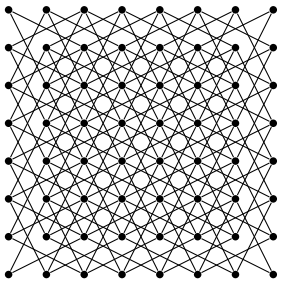
1.5 Taking photos efficiently

Here is an application of graph theory to engineering.[4] Suppose you want to inspect a manufactured part for quality by first taking photos of it from many different angles. You don't have to do this yourself: you have a many-jointed robot arm with a camera at the end, and the robot arm can move around to take the pictures for you. How can you program the robot arm to take the photos as efficiently as possible?

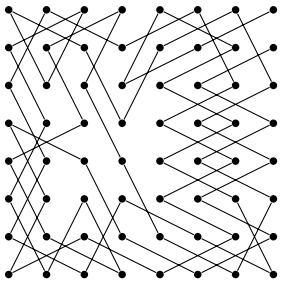
Just as in the Towers of Hanoi puzzle, we can describe this problem in terms of walks through a graph, though the setup and our goals are both different. This leads us to a good model of the problem with graph theory: if we want the robot arm's trajectory to be a sequence of vertices in a graph, then each vertex should be a position the robot arm can be in. We may as well limit the vertices to just the positions in which we want the robot arm to take a photo—those are the interesting ones.

Question:	Why not take our vertex set to be the set of all positions the robot arm can be in?
Answer:	The only reason not to is that there might be too many of these. In fact, if you imagine the robot arm moving continuously, there might be infinitely many vertices, which is definitely too many!

[4] I found it in a 2022 paper by Bottin, Boschetti, and Rosati [10], but I am not an expert in robotics, so I don't know enough to put that paper in the context of its field—I just thought it was cool.



(a) The 8×8 knight graph



(b) A Hamilton path in the graph

Figure 1.7: The knight's tour problem

Just as in the Towers of Hanoi graph, we want our edges to represent ways that the robot arm can move. However, in principle, from any position, the robot arm can twist itself around to move to any other position. There might not be a good notion of "elementary" motions: we might even end up making all pairs of vertices adjacent.

Question:	Which relevant piece of information is missing from such a model?
Answer:	The time that it takes for the robot arm to move from one position to another.

In such an application, it makes sense to consider a *weighted graph* (discussed in more detail in Chapter 9). Each edge in the graph represents a movement of the robot arm from one state to another, and it may well be that every pair of states has an edge between them. However, some motions take more time than others, so each of our edges has a nonnegative real number linked to it: the time to complete that motion. That's what a weighted graph is: each edge has a number on it that we call its *weight*, or perhaps its *cost*.

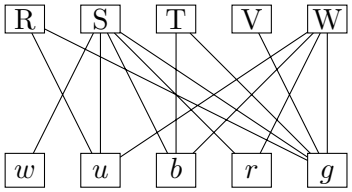
The example of the robot arm is only one instance of this problem appearing in applications: there are many situations that can be modeled by visiting all the vertices of a graph as efficiently as possible!

Question:	If we have a walk in a weighted graph, what quantity measures how good it is?
Answer:	Instead of the length of the walk, we measure its total cost (or total weight): the sum of the weights of the edges. In our application, that's the time it takes the robot to visit all the photo-taking positions and take all the photos.

An ordinary graph can be thought of as a weighted graph in which every edge has the same weight, which might as well be 1. (We can also pretend that every edge the graph doesn't have is present with a weight of ∞, so that we really really don't want to use it in an efficient solution.) In this case, a perfect solution to the problem would be a walk through the graph that visits every vertex exactly once: if there are n vertices, the walk takes $n - 1$ edges, which

	white	blue	black	red	green
Rod	3		3	2	
Sal					
Teresa	1	1		1	
Victor	1	1	3	1	
Whitney	2				

(a) Logic grid representing the puzzle



(b) Bipartite graph representing the puzzle

Figure 1.8: Two ways to think about matching cars to people

is the least number possible. In such a case, the graph has a *Hamilton path*: these are discussed in Chapter 17 of this book.

Here is another example of two very different problems becoming the same when considered from the point of view of graph theory. In chess, the "knight's tour" problem is to move a knight around the chessboard and visit each square exactly once. The graph we consider for this problem is the 8×8 knight graph, shown in Figure 1.7a: the graph with a vertex for every square of the chessboard, and edges representing the valid knight moves. A Hamilton path in this graph, shown in Figure 1.7b, is precisely a knight's tour of the chessboard.

1.6 Match the cars and drivers

A final flavor of puzzle that is secretly all about graph theory is the "logic grid puzzle", or "matching puzzle". Here is a beginner-level example.

Five friends named Rod, Sal, Teresa, Victor, and Whitney own cars in five different colors: white, blue, black, red, and green. The following three things are known about the colors of their cars:

1. Neither Victor nor Teresa own a car in a color that appears on the United States flag (red, white, or blue).
2. Even though the sounds of their names would suggest it, Rod's car is not red and Whitney's car is not white.
3. Rod and Victor like bright colors, and would not drive a black or white car.

Given this information, can you identify the color of the car each of them drives?

The classic way to solve a puzzle like this is to draw a grid to represent the data; in this case, the rows would be the 5 people in the problem, and the columns would be the 5 colors. Then, the cells of the grid representing impossible combinations are shaded in to eliminate them: in Figure 1.8a, this is shown with a number in each shaded cell indicating the statement that lets us eliminate it. From there, we can try to make deductions and use them to eliminate more cells. I have given you an example that has a unique solution, so you can try solving it, if you like.

Figure 1.8b shows another way to think about the logic puzzle: as a graph. In this graph,

- The vertices come in two types: five vertices representing the people (labeled R, S, T, V, W in the diagram) and five vertices representing the colors (labeled w, u, b, r, g in the diagram).
- There is an edge from each person to each car the three conditions permit them to drive.

This graph is called a *bipartite graph* because its vertices can be split into two non-overlapping sets such that all edges have one endpoint in each set. In this case, the two sets are the people and the cars. The solution to the puzzle is a *perfect matching* in the graph: a set of edges that have each of the vertices as an endpoint exactly once. Both of these concepts will be introduced in detail in Chapter 13.

Question:	How many edges does a perfect matching contain?
Answer:	In this problem, it will be 5 edges. In general, if our graph has n vertices, a perfect matching should have $n/2$ edges, because each of the edges has 2 endpoints, covering 2 of the n vertices.

Figure 1.8b is not necessarily the best visualization if you want to try to solve the logic puzzle. However, thinking about the problem as a graph is rewarding for two reasons:

1. We can make connections to other problems that look superficially different in how they're phrased, but turn out to be the same problem in the language of graph theory.

   Perfect matchings are not just for logic grid puzzles! On a college campus, they can be used for matching together instructors with classes, or interns with internships, or roommates in college dorms. They are also a common subroutine for more complicated optimization problems.

2. There are general graph-theoretic guarantees that apply to all these applications equally well. Later on in this book, we will see what conditions guarantee the existence of a perfect matching, and even investigate whether the solution is unique.

Question:	Suppose a graph has $n = 99$ vertices. Then a perfect matching should have $n/2$ edges, which is 49.5. How can we make sense of this?
Answer:	In a graph with 99 vertices (or any other odd number), a perfect matching cannot exist—so it's okay that our formula gives a nonsense answer. If we have 99 objects, we cannot divide them into pairs; one object will be left out at the end.

1.7 Practice problems

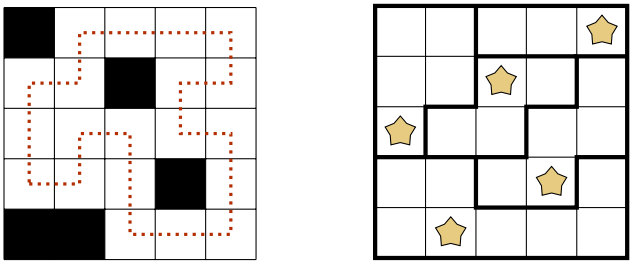
1. Describe how to use a graph to model each of the following puzzles:

   a) A word ladder puzzle, in which one word must be transformed into another by changing one letter at a time. For example, if the goal were to get from "graph" to "Euler", one possible solution might be:

   $$\text{graph} \to \text{grape} \to \text{grade} \to \text{grads} \to \text{goads} \to \text{golds} \to$$

   $$\to \text{bolds} \to \text{boles} \to \text{roles} \to \text{rules} \to \text{ruler} \to \text{Euler}.$$

   b) A pure loop puzzle, in which several squares of an $n \times n$ grid are shaded, and the goal is to draw a closed loop through the unshaded squares that visits each of them exactly once. Such a puzzle and its solution are shown in the first diagram below.



   c) A star battle puzzle, in which an $n \times n$ grid is divided into several regions, and a star must be placed in each region so that no two stars occupy the same row or column, and no two stars are adjacent even diagonally. Such a puzzle and its solution are shown in the second diagram above.

2. The country of Hungary is divided into 7 regions, which are further subdivided into 19 counties (and the capital, Budapest, which is not part of any county). Look these up on a map of Hungary; then, draw a graph diagram, similar to one of the diagrams in Figure 1.2,

   a) of the 7 regions, if you just want a bit of practice, or

   b) of the 19 counties and Budapest, if you want to draw a bigger graph.

3. Briefly explain (without checking any cases by brute force) why the cantons of Switzerland cannot be colored with just three colors so that no two adjacent cantons have the same color.

4. To study the graph representing a 3-disk Towers of Hanoi puzzle, we might label the states by 3-digit sequences 111 through 333; each digit represents the location of a disk, from largest to smallest.

   For this problem, let H_n denote the n-disk Towers of Hanoi graph; Figure 1.5 is a diagram of H_2.

   a) Draw only one part of H_3: the part containing the 9 vertices 111, 112, 113, 121, 122, 123, 131, 132, and 133.

   b) Extrapolate from what you've done to draw all of H_3.

c) How many vertices does H_n have, as a function of n?

d) How many edges does H_2 have? What about H_3? What about H_n, as a function of n?

5. The zigzag tile graph from this chapter is too large for you to draw by hand, so let's look at a much simpler problem of the same type.

   Suppose we are trying to place 2×2 square tiles on a 4×4 grid without overlap.

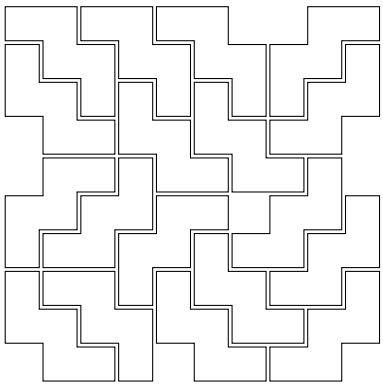
   a) Draw a diagram of the graph where the vertices are ways to place a single 2×2 tile on the grid (there should be 9 vertices), with an edge between two vertices when the tile placements they represent are incompatible.

   b) Find an independent set in the graph representing the "boring" solution, which places four 2×2 square tiles covering the entire grid.

   c) Find some other interesting non-overlapping tile placement in the grid, and find the independent set in your graph that corresponds to it.

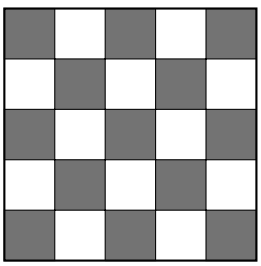
6. This is more of a puzzle than a graph-theoretical question: find a way to fit 12 zigzag-shaped tiles into an 8×8 grid with no overlaps. (How do we know that this is optimal without the use of a computer program?)

7. Find a way to color the tiling below using only 3 colors so that no two tiles that share a border have the same color.



8. A knight in chess moves by jumping to another square two steps in one direction and one step in a perpendicular direction. (Some knight moves are illustrated in Figure 1.7b.)

   a) Find a knight's tour of the 5×5 chessboard, shown below.



   b) Prove that all such tours must begin and end on a dark-colored square.

2 Isomorphisms and subgraphs

The purpose of this chapter

The definition of a graph isomorphism is sufficiently fundamental that no self-respecting graph theory book could skip it. Most authors wouldn't put this chapter so close to the beginning of the book, though.

The reason I wanted to introduce isomorphic graphs early on is for the sake of the isomorphism game in Section 2.3. By playing this game, you can discover concepts like vertex degree, subgraphs, connected components, and others naturally: that is, you come up with them because you need them for something. Later chapters will tell you the full story of these concepts, and I will then try to motivate them by telling you what they're good for—but another answer to what they're all good for is to act as graph invariants.

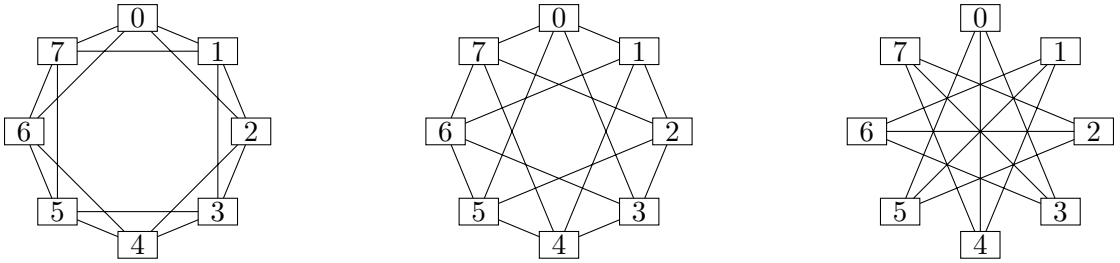
I begin with the definition of circulant graphs for several reasons. They will be useful to us many times later on, because they're a very flexible family of graphs. Their definition is also good to understand now: it will give you some practice with modular arithmetic, which will help us with other constructions in graph theory that have rotational symmetry. Finally, they give some good examples of "unexpected" graph isomorphisms, as in Proposition 2.1, and in exercises at the end of this chapter.

2.1 Circulant graphs

I have a surprising question to ask you, but before I do, I want to tell you about a family of graphs: the circulant graphs. (A set of graphs is called a "family" for the same reason that a set of people is called a family: when they're all related. But the relationship between graphs in a family is more abstract and metaphorical: they are graphs that all share an important property, or that are all defined in similar ways.)

To pick out a specific circulant graph, we need two pieces of information: an integer n (which will just be the number of vertices) and a set $\{d_1, d_2, \ldots, d_k\}$ of offsets, or jumps: integers between 1 and $n/2$. Once we have this information, the informal way to define the circulant graph is as follows: we put n vertices in a circle, and join two vertices by an edge if they are d_i steps apart in the circle, for some i. For example, Figure 2.1a shows a circulant graph with $n = 8$ vertices and with the set of jumps $\{1, 2\}$: two vertices are adjacent if they are next to each other around the circle, or two steps apart.

This is only an informal definition because we prefer not to define graphs in terms of how they're drawn: it's not forbidden to do so, but it's discouraged, because we don't want to get attached to one specific way of drawing the graph. After all, a graph is not the drawing: it is just a set of vertices and edges.



(a) $\mathrm{Ci}_8(1,2)$ (b) $\mathrm{Ci}_8(1,3)$ (c) $\mathrm{Ci}_8(3,4)$

Figure 2.1: Three circulant graphs

The mathematically precise way to describe the rules that define a circulant graph is with modular arithmetic. If you are already familiar with it, feel free to skip ahead to Definition 2.2. We will need to use modular arithmetic several times throughout this book, so let me introduce it now in a bit more detail than we need just to understand circulant graphs.

The fundamental notion of modular arithmetic is congruence modulo n, defined as follows:

Definition 2.1. *Two integers a and b are **congruent modulo** a positive integer n when a and b differ by a multiple of n: $a - b = kn$ for some k. We write this $a \equiv b \pmod{n}$.*

*For all $n \geq 1$, every integer a is congruent to exactly one element of the set $\{0, 1, \ldots, n-1\}$ modulo n; we refer to that element of $\{0, 1, \ldots, n-1\}$ as **a modulo n**, written $a \bmod n$.*

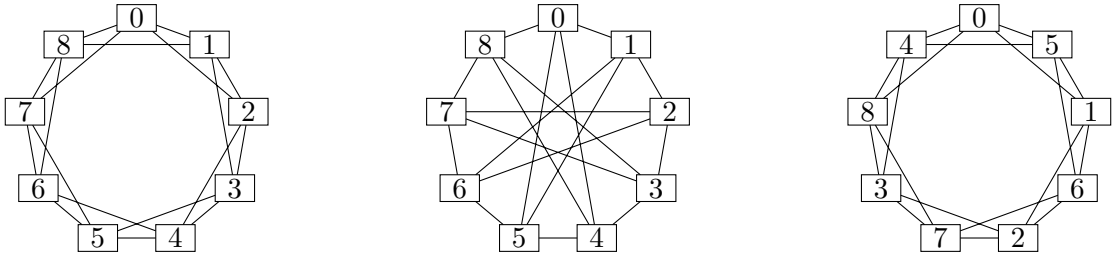
Modular arithmetic is used in situations where numbers "wrap around to the start" after n, so that $n+1$ is the same as 1, and $n+2$ is the same as 2, and so on. A common real-life situation of this type is clock arithmetic: the hours are labeled 1 through 12, but the hour after 12 is 1, not 13. The hours on a clock are placed in a circle, just as the vertices of a circulant graph, and we're going to use modular arithmetic for the same purpose!

You can treat the statement $a \equiv b \pmod{n}$ as being a relaxed kind of equality. Like equality, you can often apply the same operation to both sides to get another true statement:

- $3 \equiv 13 \pmod{10}$, so $3 + 4 \equiv 13 + 4 \pmod{10}$, or $7 \equiv 17 \pmod{10}$.
- $4 \equiv 52 \pmod{24}$, so $4 \cdot 5 \equiv 52 \cdot 5 \pmod{24}$, or $20 \equiv 260 \pmod{24}$.
- $6 \equiv -1 \pmod{7}$, so $6^2 \equiv (-1)^2 \pmod{7}$, or $36 \equiv 1 \pmod{7}$.

The one common operation you must be careful about is division. It is not always okay to go from $a \equiv b \pmod{n}$ to $\frac{a}{c} \equiv \frac{b}{c} \pmod{n}$. This is only allowed if c and n have no divisors in common! For example, we can divide both sides of $6 \equiv 36 \pmod{10}$ by 3 to get $2 \equiv 12 \pmod{10}$: a true statement. But if we divided both sides by 2, we'd get $3 \equiv 18 \pmod{10}$, which is false! (A quick way to see this: two positive integers are congruent modulo 10 if and only if their last digits are the same.)

There is much more to learn about modular arithmetic, but we've seen enough to get us through this book. So let's finally state the formal definition of a circulant graph:



(a) $\mathrm{Ci}_9(1,2)$ (b) $\mathrm{Ci}_9(1,4)$ (c) Also $\mathrm{Ci}_9(1,4)$

Figure 2.2: Comparing $\mathrm{Ci}_9(1,2)$ to $\mathrm{Ci}_9(1,4)$

Definition 2.2. *For any $n \geq 1$ and any set of integers $\{d_1, d_2, \ldots, d_k\}$ all between 1 and $n/2$, the* ***circulant graph*** $\mathbf{Ci}_n(d_1, d_2, \ldots, d_k)$ *is the graph with vertex set $\{0, 1, \ldots, n-1\}$ in which two vertices x, y are adjacent whenever $x - y \equiv \pm d_i \pmod{n}$ for some i.*

Looking back at Figure 2.1, you should check that this definition matches what you see in the diagrams, and that it matches what you'd expect from the informal definition.

Question:	Why do we say that $d_1, d_2, \ldots, d_k$ must be between 1 and $n/2$?
Answer:	Larger offsets would not mess up the definition, but they're unnecessary. If we had an offset d_i bigger than n, we could replace it by the offset $d_i \bmod n$. If we had an offset d_i between $n/2$ and n, we could replace it by $n - d_i$.

2.2 Are these the same?

The surprising question that I want to ask you now is this: are the graphs $\mathrm{Ci}_9(1,2)$ and $\mathrm{Ci}_9(1,4)$ the same? Before you answer, "No, obviously not," let me make my case.

In Figure 2.2, the first two diagrams are just the standard diagrams of $\mathrm{Ci}_9(1,2)$ and $\mathrm{Ci}_9(1,4)$. However, Figure 2.2c is something slightly different: it is also a diagram of $\mathrm{Ci}_9(1,4)$, but it has the same "shape"—the edges are drawn in all the same places—as Figure 2.2a, which shows $\mathrm{Ci}_9(1,2)$.

Question:	How do we know Figure 2.2c is also a diagram of $\mathrm{Ci}_9(1,4)$?
Answer:	For each vertex, you can check that its neighbors are the same in both Figure 2.2b and Figure 2.2c. For example, the neighbors of vertex 1 are vertices 0, 2, 5, and 6 in both diagrams.

I can't argue with the definitions, though: $\mathrm{Ci}_9(1,2)$ and $\mathrm{Ci}_9(1,4)$ are not the same graph, because they have a different set of edges. For example, $\{0,2\} \in E(\mathrm{Ci}_9(1,2))$ but $\{0,2\} \notin E(\mathrm{Ci}_9(1,4))$. And yet, Figure 2.2a and Figure 2.2c show us that the two graphs are almost identical: you can turn one into the other just by renaming the vertices! If you think about the sort of questions

we asked in Chapter 1—for example, the largest number of vertices we can select that are all adjacent—it's clear that any answer we find in $\mathrm{Ci}_9(1,2)$ will also work in $\mathrm{Ci}_9(1,4)$, and vice versa.

Question:	In $\mathrm{Ci}_9(1,4)$, the vertices 1, 5, and 6 form a triangle: the edges 15, 16, and 56 are all present. Does this correspond in any way to a triangle in $\mathrm{Ci}_9(1,2)$?
Answer:	Yes, but we have to do some "translation". Vertices 1, 5, and 6 are in the same place in Figure 2.2c as vertices 2, 1, and 3 in Figure 2.2a, so vertices 2, 1, and 3 form a triangle in $\mathrm{Ci}_9(1,2)$.

When this relationship exists between two graphs G and H—when we can turn G into H by just changing the names of the vertices—we say that G and H are isomorphic. For small graphs, this is often easy to show with a diagram, as in Figure 2.2, but it would be difficult both to draw and to check if the graphs were larger. If we had to describe how $\mathrm{Ci}_9(1,2)$ and $\mathrm{Ci}_9(1,4)$ correspond to each other without drawing the diagram, we'd want to make a list of the way we rename the vertices. For example, to turn Figure 2.2c into Figure 2.2a, we rename vertex 1 to 2, vertex 2 to 4, vertex 3 to 6, and so on. The mathematical object that describes such a correspondence is a function: a function $\varphi\{0,1,\ldots,8\} \to \{0,1,\ldots,8\}$. In general, φ should be a function from $V(G)$ to $V(H)$. Not just any function will do, however:

Definition 2.3. *An **isomorphism** from a graph G to a graph H is a function $\varphi\colon V(G) \to V(H)$ that satisfies the following properties:*

- *φ is a bijection: for every vertex $y \in V(H)$ there should be a vertex $x \in V(G)$ such that $\varphi(x) = y$. In other words, φ should have an inverse $\varphi^{-1}\colon V(H) \to V(G)$.*

  This makes the function a true correspondence, pairing each vertex in one graph with a vertex in the other.

- *φ preserves the edges: for every two vertices $x, y \in V(G)$, we have the equivalence*

$$xy \in E(G) \iff \varphi(x)\varphi(y) \in E(H).$$

  In other words, applying the function φ to go from G to H does not change which vertices are adjacent to each other.

*When there is an isomorphism from G to H, the graphs G and H are **isomorphic** to each other.*

You should not be surprised by the two properties we asked for in this definition. If you think about the things you'd do to check if the diagrams in Figure 2.2a and Figure 2.2c are the same, the two properties are just formally describing what you'd do. First, you'd check that for every vertex in one diagram, there's a vertex in the same place in the other diagram (verifying that φ is defined for all inputs, and that it's a bijection). Second, you'd want to check that every edge drawn in one diagram is also present in the other diagram (verifying that φ preserves the edges.)

Figure 2.2 suggests a particular isomorphism φ, given by the following table:

vertex of $\mathrm{Ci}_9(1,4)$	0	1	2	3	4	5	6	7	8
φ(vertex)	0	2	4	6	8	1	3	5	7

This is one possible concise way to specify an isomorphism, although for small graphs, a diagram like in Figure 2.2 is just as good. You should think of φ as being like a dictionary that translates from names in $\mathrm{Ci}_9(1,4)$ to names in $\mathrm{Ci}_9(1,2)$. In the "language" of $\mathrm{Ci}_9(1,4)$, the statement "1 is adjacent to 2, but not to 3" is true. That's a false statement about $\mathrm{Ci}_9(1,2)$, but it becomes a true one after it's been "translated"! When translated, it becomes "$\varphi(1)$ is adjacent to $\varphi(2)$, but not to $\varphi(3)$", or "2 is adjacent to 4, but not to 6".

(Just as with foreign languages, such a dictionary can have cognates: vertices like 0 that have the same role in both graphs. It can also have false cognates: both graphs have a vertex 1, but $\mathrm{Ci}_9(1,2)$'s vertex 1 does not correspond to $\mathrm{Ci}_9(1,4)$'s vertex 1.)

In practice, the best reason to work with the function φ formally is in a proof: if you have two graphs of arbitrary size given by abstract rules, and you want to prove they're isomorphic, then you can't draw a diagram—you have to give an isomorphism and verify that it works. Here's an example of such a proof:

Proposition 2.1. *For all odd integers n, the graph $\mathrm{Ci}_n(1, \frac{n-1}{2})$ is isomorphic to $\mathrm{Ci}_n(1,2)$.*

Proof. Modular arithmetic gave us the formal definition of the circulant graphs, and it can also give us the isomorphism! To find it, we need to find a pattern in the $n = 9$ case (where we already have an isomorphism) and generalize.

Question:	Is there a pattern in the table defining the isomorphism φ from $\mathrm{Ci}_9(1,4)$ to $\mathrm{Ci}_9(1,2)$?
Answer:	Yes; probably the first one you will spot is that the values of φ in the table begin by going up by 2's from left to right.

In fact, we can check that the formula $\varphi(x) = 2x \bmod 9$ works for every value in the table, and so to prove the generalization we want, we can try defining $\varphi(x) = 2x \bmod n$.

To check that φ is a bijection, we can find an inverse for it. In this case, there happens to be an inverse with a short formula: $\varphi^{-1}(x) = \frac{n+1}{2} \cdot x \bmod n$. We can check that this really is an inverse, because

$$\frac{n+1}{2} \cdot 2x = (n+1)x \equiv x \pmod{n}.$$

When working modulo n, multiplication by $\frac{n+1}{2}$ undoes multiplication by 2, and vice versa. This is usually the best way to proceed: it's possible to check that φ is a bijection by checking that it's injective and surjective (one-to-one and onto), but this doesn't usually save us any work, and the inverse might be useful for us.

Next, we check that φ maps vertices adjacent in $\mathrm{Ci}_9(1, \frac{n-1}{2})$ to vertices in $\mathrm{Ci}_9(1,2)$. If two vertices x and y are adjacent in $\mathrm{Ci}_9(1,4)$, it could be for four reasons; we could have $x - y \equiv 1 \pmod{n}$, or $x - y \equiv -1 \pmod{n}$, or $x - y \equiv \frac{n-1}{2} \pmod{n}$, or $x - y \equiv -\frac{n-1}{2} \pmod{n}$.

- If $x - y \equiv 1 \pmod{n}$, then $\varphi(x) - \varphi(y) = 2(x - y) \equiv 2 \pmod{n}$.
- If $x - y \equiv -1 \pmod{n}$, then $\varphi(x) - \varphi(y) = 2(x - y) \equiv -2 \pmod{n}$.
- If $x - y \equiv \frac{n-1}{2} \pmod{n}$, then $\varphi(x) - \varphi(y) = 2(x - y) \equiv 2(\frac{n-1}{2}) = n - 1 \equiv -1 \pmod{n}$.
- Finally, if $x - y \equiv -\frac{n-1}{2} \pmod{n}$, then $\varphi(x) - \varphi(y) = 2(x - y) \equiv 2(-\frac{n-1}{2})$, which simplifies to $1 - n$, and $1 - n \equiv 1 \pmod{n}$.

We have an if-and-only-if statement, so normally the next step would be to check the converse: either that φ sends non-adjacent vertices in $\mathrm{Ci}_n(1, \frac{n-1}{2})$ to non-adjacent vertices in $\mathrm{Ci}_n(1, 2)$, or that φ^{-1} sends adjacent vertices in $\mathrm{Ci}_n(1, 2)$ to adjacent vertices in $\mathrm{Ci}_n(1, \frac{n-1}{2})$. In this case, however, we can skip that step by doing some counting instead.

Both $\mathrm{Ci}_n(1, \frac{n-1}{2})$ and $\mathrm{Ci}_n(1, 2)$ both have $2n$ edges: each one is a circulant graph with two offsets, and for each offset d_i, there are n pairs x, y such that $x - y \equiv d_i \pmod{n}$.

Question:	Is it always true for circulant graphs that the number of offsets tells you the number of edges?
Answer:	Almost always, with one exception: when n is even, the offset $\frac{n}{2}$ only produces half the edges you'd expect. This is because $x - y \equiv \frac{n}{2} \pmod{n}$ and $x - y \equiv -\frac{n}{2} \pmod{n}$ are the same condition! Intuitively, going $\frac{n}{2}$ steps left around the circle is the same as going $\frac{n}{2}$ steps right when n is even: both go to the opposite vertex on the circle.

We've already checked that φ is a bijection that sends each of the $2n$ edges of $\mathrm{Ci}_n(1, \frac{n-1}{2})$ to an edge of $\mathrm{Ci}_n(1, 2)$. This "uses up" all $2n$ edges of $\mathrm{Ci}_n(1, 2)$. So for a pair of vertices that are not adjacent in $\mathrm{Ci}_n(1, \frac{n-1}{2})$, their images cannot be adjacent in $\mathrm{Ci}_n(1, 2)$, verifying the converse.

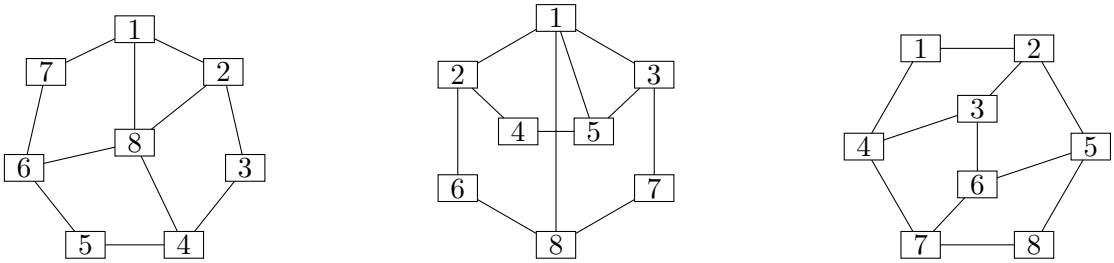
This completes the proof that φ is an isomorphism from $\mathrm{Ci}_n(1, \frac{n-1}{2})$ to $\mathrm{Ci}_n(1, 2)$, showing also that the two graphs are isomorphic. □

An isomorphism from G to H reveals a way in which G is the same as H, so an isomorphism from a graph G to G itself reveals a way in which G is the same as G. This doesn't seem useful at first: we already know that G is the same as G. However, it is useful: it reveals symmetry in the graph. For example, one of the main reasons circulant graphs matter in graph theory is their cyclic symmetry: rotating one of the diagrams in Figure 2.1 by 45°, or one of the graphs in Figure 2.2 by 40°, doesn't change the shape of the diagram. To state this precisely, we need to discuss automorphisms.

Definition 2.4. *An **automorphism** of a graph G is an isomorphism from G to itself.*

To describe the symmetry of an n-vertex circulant graph, we might say that the function α which rotates its standard diagram by $\frac{360°}{n}$ is an automorphism.

Question:	What is the definition of α using modular arithmetic?
Answer:	It is the function from $\{0, 1, \ldots, n-1\}$ to $\{0, 1, \ldots, n-1\}$ given by $\alpha(x) = x + 1 \bmod n$.



(a) The graph G_1 (b) The graph G_2 (c) The graph G_3

Figure 2.3: A three-graph set for the isomorphism game

We can also spot a different kind of symmetry in the diagrams of our circulant graphs: if you reflect them, swapping left and right, the shape does not change. This corresponds to a different isomorphism $\beta\colon \{0, 1, \ldots, n-1\} \to \{0, 1, \ldots, n-1\}$, where $\beta(x) = -x \bmod n$.

Different diagrams can make different automorphisms easier to see, although the automorphism does not depend on the diagram: α and β are automorphisms of a circulant graph directly from the definition, regardless of how we draw it. More complicated symmetries might not even be possible to illustrate in a diagram!

Even if a graph G has no symmetries to speak of, the function $\varphi\colon V(G) \to V(G)$ defined by $\varphi(x) = x$ for all $x \in V(G)$ is an automorphism of G. This is called the *identity automorphism* or *trivial automorphism*, and is much less exciting to discover. (It has a name mostly so that we can talk about automorphisms other than the trivial one, or about graphs that do or don't have nontrivial automorphisms.)

2.3 The isomorphism game

How do we find an isomorphism between two graphs when their diagrams don't completely line up? And when an isomorphism doesn't exist, how can we tell? We don't want to use brute force: with two n-vertex graphs, there are $n!$ possible bijections, which is too many to check by hand even for fairly small n.

To teach you how to do it, I will teach you to play the "isomorphism game". In this game, I show you three graphs. Two of them are isomorphic to each other, and the third is different. The goal is to identify which graph is the odd one out, and to find an isomorphism between the other two. Let me show you how the game is played on an example.

In Figure 2.3, if we consider the possibility that G_1 and G_2 are isomorphic, how could we begin to find the isomorphism? A good start is to look for a distinguishing feature of some of the vertices. Be careful, though! Something like, "In G_1, vertex 8 is all by itself in the middle, unlike the others, which are arranged in a circle" is not a distinguishing feature of vertex 8's role in the graph, but merely in the diagram; it's not useful.

We look, instead, for distinguishing features that can't disappear when we rearrange the vertices and relabel them. For example, vertex 8 of G_1 has four neighbors: 1, 2, 4, and 6. An isomorphism $\varphi\colon V(G_1) \to V(G_2)$ must preserve the edges $18, 28, 48, 68$: the pairs $\varphi(1)\varphi(8)$, $\varphi(2)\varphi(8)$, $\varphi(4)\varphi(8)$, and $\varphi(6)\varphi(8)$ must be edges of G_2. So $\varphi(8)$ is a vertex of G_2 with four

neighbors: $\varphi(1)$, $\varphi(2)$, $\varphi(4)$, and $\varphi(8)$. Which vertex can that be? It can only be vertex 1. So we can deduce that if the isomorphism φ exists, then $\varphi(8) = 1$.

This has given us a foothold, and now we can build on it: to a complete isomorphism, or to a contradiction. Vertices $1, 2, 4, 6$ (the neighbors of 8) in G_1 must be sent to vertices $2, 3, 5, 8$ in G_2: the neighbors of 1, which is $\varphi(8)$. But can we distinguish them further? Well, in G_1, 1 and 2 are also adjacent to each other, so $\varphi(1)$ and $\varphi(2)$ must be adjacent. The only adjacent pair of neighbors that vertex 1 has in G_2 is the pair $\{3, 5\}$. So $\{\varphi(1), \varphi(2)\}$ must be equal to $\{3, 5\}$, in some order.

We don't like the words "in some order"—that way lies brute-force checking of cases. But in this case, there's an explanation for it. Do you see that in Figure 2.3a, if we draw a straight line passing through vertices 5 and 8, then it is a line of symmetry of the diagram? That symmetry gives us an automorphism of G_1, and that automorphism swaps vertices 1 and 2. This means that vertices 1 and 2 play identical roles, as far as isomorphisms go: anything one of them can do, the other can do just as well. We can arbitrarily decide $\varphi(1) = 3$ and $\varphi(2) = 5$ without fear of an error. (See how useful automorphisms can be, after all?)

At this point, the rest of φ can be determined. The neighbors of 2 in G_1 are 1, 3, and 8. The neighbors of $5 = \varphi(2)$ in G_2 are $1 = \varphi(8)$, $3 = \varphi(1)$, and 4. Two out of three neighbors have already been matched up by φ, so the remaining pair must also go together: $\varphi(3) = 4$. Going around the circle, we can find where φ sends $4, 5, 6, 7$ and get the following isomorphism:

vertex of G	1	2	3	4	5	6	7	8
φ(vertex)	3	5	4	2	6	8	7	1

We're halfway done with the game. How do we play the other half, and explain why G_1 and G_3 are not isomorphic?

The argument begins with the same way; we look for a distinguishing feature. As before, if $\psi\colon V(G_1) \to V(G_3)$ is an isomorphism, then $\psi(8)$ must be a vertex of G_3 with four neighbors: $\psi(1)$, $\psi(2)$, $\psi(4)$, and $\psi(8)$. Which vertex can that be? Well, there is no such vertex! In G_3, vertices 1 and 8 have two neighbors, and the rest each have three neighbors; no vertex has four. So ψ cannot exist: G_1 and G_3 are not isomorphic.

Question:	Is it correct to say that G_1 is not isomorphic to G_3 because vertex 1 has three neighbors in G_1, but only two neighbors in G_3?
Answer:	No: an isomorphism is allowed to change which vertex is "vertex 1".

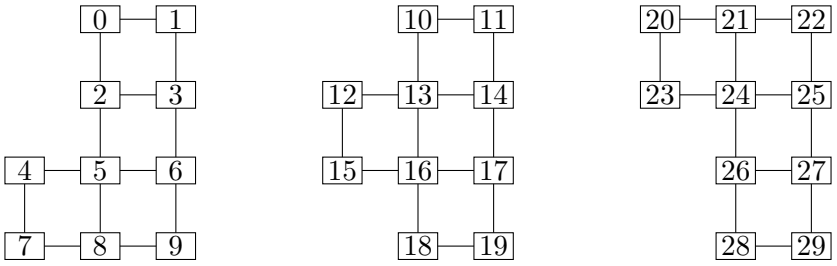
Question:	Is it correct to say that G_1 is not isomorphic to G_3 because G_1 has three vertices (3, 5, and 7) with two neighbors each, but G_2 has only two such vertices (1 and 8)?
Answer:	Yes: if there were an isomorphism $\psi\colon V(G_1) \to V(G_3)$, then $\psi(3)$, $\psi(5)$, and $\psi(7)$ would be three different vertices with two neighbors each. But there are not three such vertices in G_3.

Question:	Is it correct to say that G_1 is isomorphic to G_2 just because for every number k, the two graphs have the same number of vertices with k neighbors?
Answer:	No: we must find the isomorphism before we draw this conclusion. It's possible for two graphs to agree in this way, and yet not be isomorphic, and you'll see examples of this on the next page.

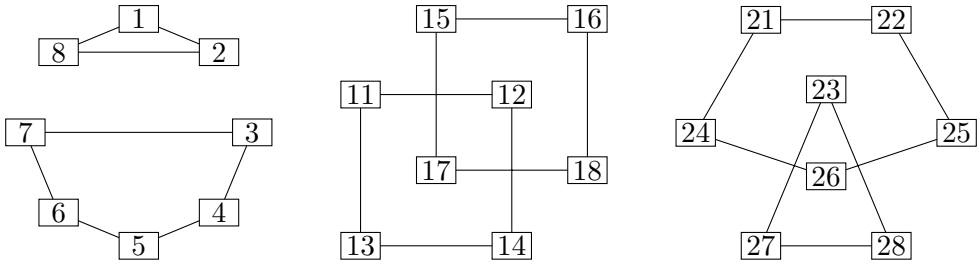
In general, to prove that two graphs are not isomorphic, it's enough to find any difference between the two graphs that any isomorphism must preserve: something that's inherent to the structure of the graph, and not a feature of the vertex labels or the way the graph is drawn.

Before telling you more about such properties, I will let you discover some of them for yourself as you play the isomorphism game. (Try doing this yourself before you look at any spoilers.)

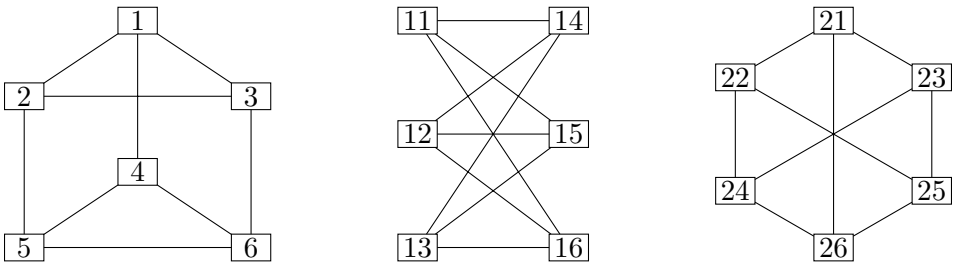
Problem 2.1. *Determine which of the three graphs below is the odd one out, and find an isomorphism between the other two.*



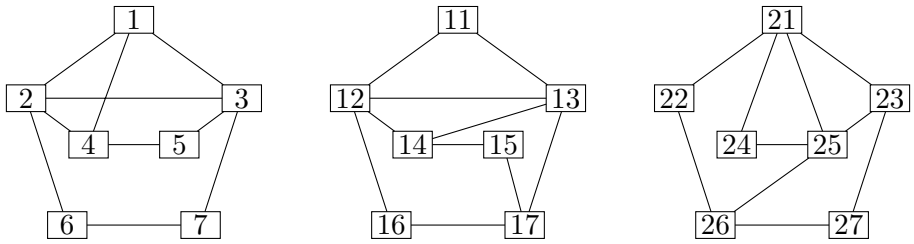
Problem 2.2. *Determine which of the three graphs below is the odd one out, and find an isomorphism between the other two.*



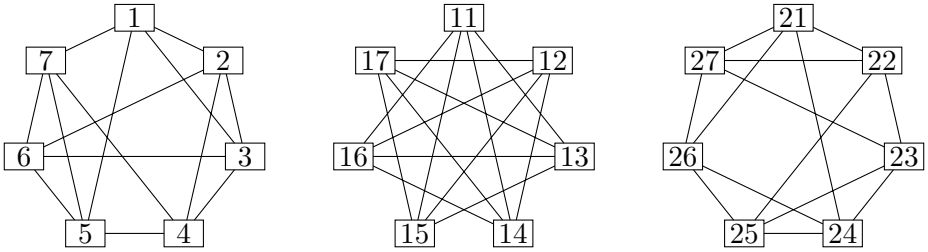
Problem 2.3. *Determine which of the three graphs below is the odd one out, and find an isomorphism between the other two.*



Problem 2.4. *Determine which of the three graphs below is the odd one out, and find an isomorphism between the other two.*



Problem 2.5. *Determine which of the three graphs below is the odd one out, and find an isomorphism between the other two.*



2.4 Graph invariants

When playing the isomorphism game, you may say things like, "These two graphs are not the same as that other graph, because these two graphs both X, and that other graph does not X. But a graph that X can't be isomorphic to a graph that does not X." Whenever you can say that, X is called a graph invariant:

Definition 2.5. *A **graph invariant** is any property of a graph (a result of any kind that can be computed from the graph) such that, whenever two graphs G and H are isomorphic, they must agree in that property.*

Graph invariants can be binary (true or false), or numerical (like the number of vertices or edges), or even more complicated, such as a sequence of numbers. They're called "invariants" because don't vary: they can't be changed by drawing the diagram differently, or by relabeling the vertices. In other words, two isomorphic graphs have to agree on all graph invariants.

Sometimes graph invariants are also called "graph properties". Using the word "graph property" is also taking a philosophical stance: it's implying that the only true graph properties—the only properties of graphs that are worth studying in the discipline of graph theory—are graph invariants! I think this is true, though I will carve one or two exceptions out for myself in a couple of pages.

Of course, as mathematicians we can't just intuit our way into claiming something is a graph invariant: we have to prove it. I will show you two of these proofs, one for a numerical invariant

and one for a true/false invariant. Lots of these proofs are very similar to each other, and become boring once you get the hang of them, but I encourage you to try writing one or two yourself until you get to the point where you can see how you'd go about writing it before you even start.

Proposition 2.2. *The number of vertices and the number of edges of a graph are both graph invariants.*[5]

Proof. In combinatorics, the classical way to prove that two sets are equal in size is to find a bijection between them. So to prove this theorem, we want to show that if two graphs G and H are isomorphic, then there's a bijection between $V(G)$ and $V(H)$, as well as a bijection between $E(G)$ and $E(H)$.

One of these proofs is very short! By definition, an isomorphism from G to H is a bijection $\varphi\colon V(G) \to V(H)$, which is exactly what we needed, and which automatically means that $|V(G)| = |V(H)|$. We conclude that the number of vertices is a graph property.

For the number of edges, we need to work harder: we need to use φ to construct a bijection $E(G) \to E(H)$, which we'll call φ'. For an edge $xy \in E(G)$, we'll define $\varphi'(xy)$ to be the edge $\varphi(x)\varphi(y)$. There are two checks necessary to make sure that this is legitimate:

- $\varphi'(xy)$ really is an element of $E(H)$. That's because φ is a graph isomorphism, so it preserves edges: x and y are adjacent, so $\varphi(x)$ and $\varphi(y)$ must be adjacent, which means that $\varphi(x)\varphi(y)$ is an edge of H.
- The edge xy also goes by a different name: yx is the same edge. We want to make sure φ' does the same thing to it when it's going by a different name. Fortunately, $\varphi'(yx) = \varphi(y)\varphi(x)$ is another name for $\varphi(x)\varphi(y) = \varphi'(xy)$.

Next, we check that φ' is a bijection. One way to do this is to construct an inverse for it, and that's convenient to do here because we already know that φ has an inverse $\varphi^{-1}\colon V(H) \to V(G)$. So define $\varphi'^{-1}(xy) = \varphi^{-1}(x)\varphi^{-1}(y)$ for every edge $xy \in E(H)$. As before, there are two checks to make sure that this is legitimate, but they are the same as for φ'. Finally, we want to check that φ' and φ'^{-1} are inverses. For an edge $xy \in E(G)$, $\varphi'^{-1}(\varphi'(xy))$ simplifies to $\varphi^{-1}(\varphi(x))\varphi^{-1}(\varphi(y))$ or just xy; similarly, for an edge $xy \in E(H)$, $\varphi'(\varphi'^{-1}(xy))$ simplifies to xy.

This proves that φ' and φ'^{-1} are inverses, so we conclude that φ' is a bijection; this is exactly what we needed to know that $|E(G)| = |E(H)|$, and that completes the proof that the number of edges in a graph is a graph invariant. □

For the next example, I will give you a graph invariant that feels ad-hoc and doesn't have a name, just to make the point that something ad-hoc with no name can still be a graph invariant.

Proposition 2.3. *The property of having a vertex adjacent to all other vertices is a graph invariant.*

[5] Some people like to say that the number of vertices in a graph is its order, and the number of edges is the graph's size. I won't use this terminology, because it feels too arbitrary, and it's not like "number of vertices" takes too long to say.

Proof. Let G and H be two isomorphic graphs, and let $\varphi\colon V(G) \to V(H)$ be an isomorphism between them. Suppose that G has a vertex x adjacent to all other vertices of G; then to complete our proof, we must show that H has such a vertex, too.

Well, vertex x in G corresponds to vertex $\varphi(x)$ in H, so it's natural to suppose that $\varphi(x)$ is the vertex of H adjacent to all others. Now we must prove it. To do so, let y be an arbitrary vertex of H not equal to $\varphi(x)$.

Because φ is an isomorphism, it's a bijection, and has an inverse $\varphi^{-1}\colon V(H) \to V(H)$. So G has a vertex $\varphi^{-1}(y)$; we know that $\varphi^{-1}(y) \neq x$ because we picked y to be different from $\varphi(x)$.

Since x is adjacent to all other vertices of G, and $\varphi^{-1}(y)$ is one of them, in particular x is adjacent to $\varphi^{-1}(y)$. Because φ is an isomorphism, it preserves adjacency, which means that $\varphi(x)$ is adjacent to $\varphi(\varphi^{-1}(y)) = y$. We chose y to be an arbitrary vertex of H other than $\varphi(x)$, so this proves that $\varphi(x)$ is adjacent to all such vertices, completing the proof of the theorem. □

Let me discuss some of the other graph invariants you may or may not have discovered in the course of playing the isomorphism game.

Whether a graph is *connected* is a graph invariant: if it's not, the number of *connected components* is a graph invariant. (Two vertices are in the same component if you can get from one to the other by following edges; more on this in Chapter 3.)

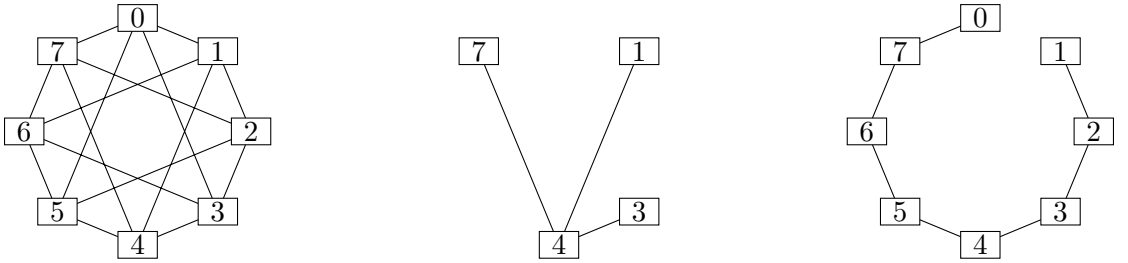
Whether a graph has a "piece" of a certain form is also an invariant; to make this formal, we will need the idea of subgraphs, which are discussed in the final section of this chapter.

If we're careful, we can get a graph invariant out of vertex degrees: the *degree* of a vertex is the number of edges incident to it. (We'll discuss vertex degrees in more detail in Chapter 4.)

We have to be careful because something like "the degree of vertex x" is not an invariant: an isomorphism can change which vertex is x. But the binary property "there exists a vertex of degree n" is invariant, and so is "the number of vertices of degree n".

The ultimate way to track the vertex degrees would be a tally of how many vertices have each degree, such as for example "5 vertices of degree 2, 4 vertices of degree 3, and 1 vertex of degree 4" for the first graph in Problem 2.1. It would be reasonable to call this a "degree tally", but the convention instead is to sort the degrees from largest to smallest (such as $4, 3, 3, 3, 3, 2, 2, 2, 2, 2$) and call this the *degree sequence*. (See Chapter 5 and Chapter 6 for more details.)

Here is a subtle example of a graph property: we say that a graph is *planar* if it can be drawn in the plane without any edges crossing, and whether or not a graph is planar is a graph invariant. This does not mean that "do any of the edges cross?" is a graph invariant—it's not, because it depends on the way the graph is drawn! But the potential to have a crossing-free drawing is something that cannot be taken away with an isomorphism. (See Chapter 21 for more details.) The subtlety in the previous paragraph is one of the exceptions that I wanted to mention about what does and doesn't count as an invariant.



(a) $\mathrm{Ci}_8(1,3)$ (b) An induced subgraph (c) A spanning subgraph

Figure 2.4: Two subgraphs of a circulant graph

2.5 Subgraphs and some common small graphs

Some of the graph invariants you may have made use of involve finding smaller graphs that appear inside bigger ones. We introduce subgraphs to make this idea precise.

Figure 2.4 shows the circulant graph $\mathrm{Ci}_8(1,3)$ and some of its subgraphs.

Definition 2.6. *A **subgraph** H of a graph G is a graph that includes some of G's vertices and some of G's edges: $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$.*

Keep in mind that we cannot include an edge xy in the subgraph unless we have included vertices x and y; otherwise, that edge just wouldn't make sense.

There are two special types of subgraphs that deserve special mention. First:

Definition 2.7. *A **spanning subgraph** H of a graph G is a subgraph such that $V(G) = V(H)$.*

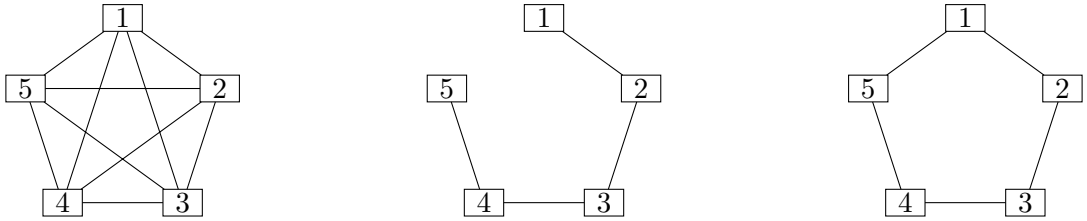
There are no requirements to include any edges: one possible spanning subgraph is the subgraph with all of G's vertices, but no edges at all. Figure 2.4c shows a spanning subgraph of $\mathrm{Ci}_8(1,3)$.

Definition 2.8. *An **induced subgraph** H of a graph G is a subgraph that includes all the edges it possibly could: for all edges $xy \in E(G)$ such that $x \in V(H)$ and $y \in V(H)$, it is true that $xy \in E(H)$. The **subgraph of G induced by S**, written $G[S]$, is the unique induced subgraph of G with vertex set S, where S is a subset of $V(G)$.*

Figure 2.4b shows an example: the subgraph of $\mathrm{Ci}_8(1,3)$ induced by the set $\{1,3,4,7\}$.

It's also common to define a subgraph that contains almost of G by deleting vertices or edges. To delete a single edge: if $xy \in E(G)$, then we write $G - xy$ for the subgraph including all vertices of G and all edges except edge xy. To delete a single vertex: if $x \in V(G)$, then we write $G - x$ for the subgraph including all vertices of G except x, and all edges except those incident to x. (Those have to go; they have no meaning if x is no longer a vertex.) More generally, if S is a set of vertices or of edges, we write $G - S$ for the subgraph where all elements of S, and all edges with an endpoint in S, are deleted. For example, Figure 2.4b could also be described as $\mathrm{Ci}_8(1,3) - \{0,2,5,6\}$.

We can get graph invariants by looking at subgraphs of various graphs isomorphic to some fixed graph H. Some terminology helps us concisely refer to this situation, which is very common (especially when H is small):



(a) The graph K_5 (b) The graph P_5 (c) The graph C_5

Figure 2.5: Several common graphs

Definition 2.9. *A **copy of H** is a graph isomorphic to H; we often say that a graph G **contains a copy of H** if G has a subgraph isomorphic to H.*

For any graph H, the true/false property "contains a copy of H" and the numerical value "the number of copies of H" are both graph invariants.

There are several families of graphs that are very commonly encountered, and so they're given names that any graph theorist would recognize. They are useful subgraphs to look for in a graph, as well. I will define three of them in this chapter:

Definition 2.10. *For any $n \geq 1$, the **complete graph K_n** is the graph with vertex set $\{1, \ldots, n\}$ and all $\binom{n}{2}$ possible edges $\{i, j\}$ where $1 \leq i < j \leq n$.*

A copy of K_n in a graph is often called a *clique*, though when I introduce this term officially in Chapter 18, I will give it a related but slightly different meaning.

Definition 2.11. *For any $n \geq 1$, the **path graph P_n** is the graph with vertex set $\{1, \ldots, n\}$ and $n-1$ edges: the edges $\{i, i+1\}$ where $1 \leq i \leq n-1$. A copy of P_n (that is, a graph isomorphic to P_n) is simply called a **path**.*

Definition 2.12. *For any $n \geq 3$, the **cycle graph C_n** has all the vertices and edges of P_n, plus one additional edge: the edge $\{1, n\}$. A copy of C_n is simply called a **cycle**.*

In the next chapter, we will learn more about paths and cycles.

Question:	Why does our definition of C_n require $n \geq 3$?
Answer:	For $n = 1$ and $n = 2$, adding the edge $\{1, n\}$ to P_n doesn't make sense. When $n = 1$, we'd be adding the set $\{1, 1\}$, which is not an edge, and when $n = 2$, the edge $\{1, 2\}$ would already be present in P_2. In Chapter 7, we will encounter two *multigraphs* that could be called C_1 and C_2.

Figure 2.5 includes an example of each of these families. I should mention that while every graph theorist will draw the same unlabeled diagram of each graph, there is no agreement on what the vertex sets $V(K_n)$, $V(P_n)$, and $V(C_n)$ are. I have chosen to use the set $\{1, 2, \ldots, n\}$ in all three cases, because in my opinion, that's a good concrete default.

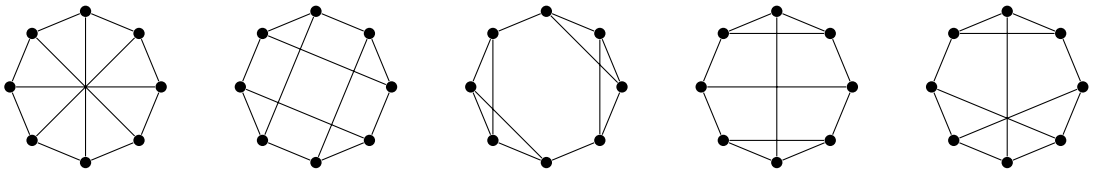
The reason there is no agreement about the vertex sets among graph theorists is because the exact vertex set is hardly ever relevant. If you want to answer questions like, "What is the number of copies of C_4 in a graph G?" then your answer will be the same no matter what names you gave to the vertices of C_4.

Similarly, the graphs $\mathrm{Ci}_n(1)$ and C_n are isomorphic, but not equal by the definitions we gave: the vertices of $\mathrm{Ci}_n(1)$ are $\{0, 1, \ldots, n-1\}$ (which helps with the modular arithmetic) while the vertices of C_n are $\{1, 2, \ldots, n\}$. However, we are often fine saying that "$\mathrm{Ci}_n(1)$ is a cycle graph" and "C_n is a circulant graph", because the important thing about the definition of these graphs is their structure, not the vertex set.

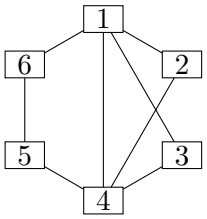
Question:	Is K_n a circulant graph, in this sense?
Answer:	Yes: it is isomorphic to $\mathrm{Ci}_n(1, 2, \ldots, \frac{n-1}{2})$ or $\mathrm{Ci}_n(1, 2, \ldots, \frac{n}{2})$, depending on whether n is odd or even: a circulant graph with all the possible offsets, in either case.

2.6 Practice problems

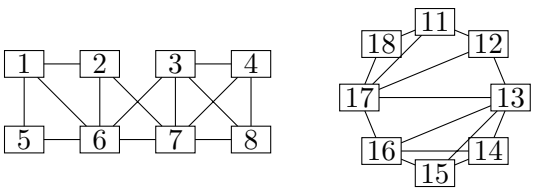
1. Prove that none of these five graphs are isomorphic: find invariants distinguishing them all from each other.



2. Find 11 different graphs with 4 vertices so that no two of the graphs are isomorphic.
3. In each of the circulant graphs in Figure 2.1, count the number of copies of C_5.
4. Find all four automorphisms of the graph shown below.



5. Find two different isomorphisms between the two graphs below:



6. a) How many induced subgraphs with at least one vertex does the complete graph K_4 have?

   b) How many spanning subgraphs does K_4 have?

   c) How many subgraphs of any kind (but with at least one vertex) does K_4 have? This one is trickier; you will need to think about the structure of K_4.

7. Let G and H be isomorphic graphs. Prove the following:

   a) G and H have the same number of vertices of degree 4.

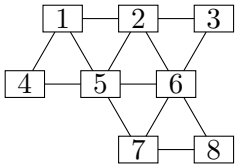
   b) If G contains a copy of K_3, then H contains a copy of K_3.

8. Show that the circulant graphs $\mathrm{Ci}_{13}(1, 3, 4)$ and $\mathrm{Ci}_{13}(2, 5, 6)$ are isomorphic.

9. When an automorphism of G takes a vertex x to another vertex y, then we say that x and y are similar.

   a) Prove that if x and y are similar, then $G - x$ and $G - y$ are isomorphic.

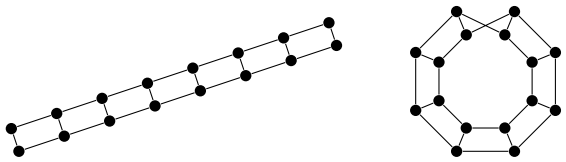
   On the other hand, the converse to (a) is false! Let G be the graph below:



   b) Prove that $G - 4$ is isomorphic to $G - 8$.

   c) Prove that 4 and 8 are not similar. (In fact, G has no isomorphisms other than the identity automorphism.)

10. The n-vertex Möbius ladder graph is defined for every even $n > 4$. It has that name because it resembles a Möbius strip, which can be made out of a long strip of paper by taping the ends together, with a twist. Similarly, the Möbius ladder graph is obtained by taking a "long strip" (two copies of $P_{n/2}$ with edges between their corresponding vertices, as shown below on the left) and joining the ends together with a twist (as shown below on the right).



   a) Give a formal definition of the Möbius ladder graph based on the diagram above, naming the vertices however you like.

   b) Prove that for all even $n > 4$, the n-vertex Möbius ladder graph is isomorphic to the circulant graph $\mathrm{Ci}_n(1, \frac{n}{2})$.

3 Walks, paths, and cycles

The purpose of this chapter

Without learning about walks in a graph first, we would find it hard to study almost anything else. Do you want to understand regular graphs (Chapter 5)? Cycles are the quintessential regular graph. Euler tours (Chapter 8)? A kind of closed walk. Trees (Chapter 9)? They are connected, and they have no cycles, both properties that come back to this chapter. Matchings (Chapter 13)? Understanding augmenting paths is key to finding larger matchings. I could go on, but I already have a section devoted to topics that this chapter will help you understand: the table of contents.

This chapter is one of the first in which different will give you slightly different definitions of the concepts we introduce. I have chosen to define paths and cycles as subgraphs which are represented by, but not the same as, walks. Other authors say that paths and cycles are a type of walk, and still other authors use the word "path" to mean "walk" and say "simple path" instead of "path". In my opinion, the terminology I've picked is the terminology that best reflects actual usage by mathematicians: no matter how we define paths and cycles, we treat them as subgraphs, count them as subgraphs, and do things with them that can only be done with graphs. By saying that a walk *represents* a path or cycle, I also help us remember that there can be multiple such representations.

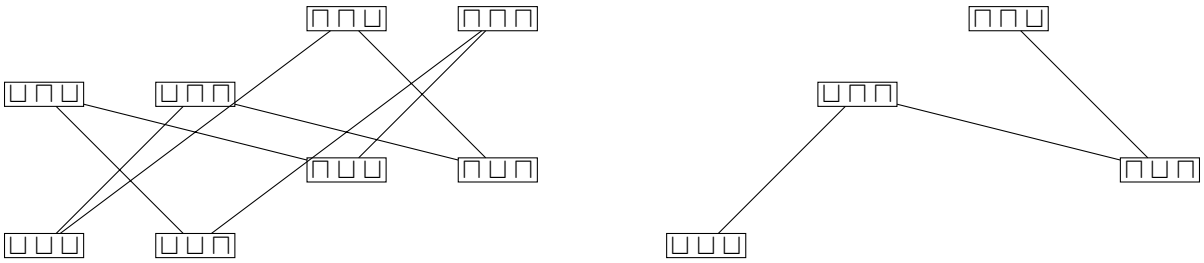
Anyway, justification aside, this also goes to warn you that in graph theory, not every source you read will give you the same definitions. The definitions should be interchangeable, once you learn how. After all, all graph theorists are still doing the same math, no matter how we talk about it.

I include the proof of Theorem 3.1 in this chapter, because it would be strange to postpone the proof to an appendix; however, the natural place to study the proof is in Appendix A, because it is a good early example of a proof by the extremal principle. Similarly, Lemma 3.3 is a good example of unpacking definitions, which I also discuss in Appendix A.

3.1 Walks and paths

Last time we looked at walks in a graph, it was in the context of the Towers of Hanoi puzzle, in Chapter 1; this time, let's look at a simpler puzzle.

To set up this puzzle, you will need three empty cups; ideally, they are empty mugs of beer, because this puzzle is not difficult enough to challenge anyone sober for very long. They are placed in a row on the table, right side up: ⊔⊔⊔. A valid move in this puzzle is to take two cups that are next to each other—the first two, or the last two—and flip them over. From the starting state, a single move can take us to the ⊓⊓⊔ state if the first two cups are flipped, or



(a) The state graph of the three cup puzzle

(b) Not a solution

Figure 3.1: The three cup puzzle

to the ⊔⊓⊓ state if the last two cups are flipped. The goal of the puzzle is to flip all three cups upside down.[6]

Figure 3.1a shows a graph of all the states possible in the three cup puzzle, where two vertices are adjacent if it's possible to move from one state to the other. (This is a symmetric relation, because all valid moves are reversible.)

In Chapter 2, we defined paths in a graph G: copies of the path graph P_n, for some n. A big part of the reason why we define paths is exactly to describe situations like the three cup puzzle. For example, the subgraph shown in Figure 3.1b is a 4-vertex path. By following the edges of the path, we can get from ⊔⊔⊔ to ⊓⊓⊔. If only we could find a different subgraph: a path which included vertices ⊓⊔⊓ and ⊓⊓⊓. This would tell us how to solve the impossible puzzle!

It's frequently useful to summarize a path by listing all the vertices along it from one end to the other; for example, for the path in Figure 3.1b, we could write down the sequence

$$(⊔⊔⊔,\ ⊔⊓⊓,\ ⊓⊔⊓,\ ⊓⊓⊔).$$

In general, for a subgraph P isomorphic to P_n by the isomorphism $\varphi\colon V(P_n) \to V(P)$, we could write the sequence $(\varphi(1), \varphi(2), \ldots, \varphi(n))$. In such a sequence, there is an edge between each pair of consecutive vertices.

There are other such sequences, that do not represent paths, and if you find a suitable victim for the three cups puzzle, you will discover many of them by writing down your victim's fruitless attempts. For example, the sequence

$$(⊔⊔⊔,\ ⊔⊓⊓,\ ⊓⊔⊓,\ ⊓⊓⊔,\ ⊓⊔⊓)$$

also has the property that there is an edge between each pair of consecutive vertices, but it does not represent a path.

[6] To make the puzzle appear possible to solve, you can do the following: set up the puzzle in the alternating ⊔⊓⊔ state, and make a few moves back and forth before arriving at the desired ⊓⊓⊓. Then explain that you're "resetting" the puzzle, and bring it back to an alternating state before letting your victim try solving it—but make it, instead, the ⊓⊔⊓ state. After this, the puzzle will be impossible. You can further disguise the nature of the puzzle by using 5 cups instead of 3.

Question:	Why doesn't it represent a path?
Answer:	In a sequence of the form $(\varphi(1), \varphi(2), \ldots, \varphi(n))$, all n vertices would be different, because φ is an isomorphism and therefore a bijection. Here, some vertices do repeat.

Let's make some definitions to generalize this idea:

Definition 3.1. *A **walk** in a graph G is a sequence $(x_0, x_1, \ldots, x_l)$ of vertices of G, such that for each $i = 1, \ldots, l$, vertices x_{i-1} and x_i are adjacent: $x_{i-1}x_i \in E(G)$. A walk whose first vertex is x and whose last vertex is y is an **$x - y$ walk**.*

*A walk $(x_0, x_1, \ldots, x_l)$ **represents** a path P in G when $V(P) = \{x_0, x_1, \ldots, x_l\}$ and $E(P) = \{x_0x_1, x_1x_2, \ldots, x_{l-1}x_l\}$; a walk represents a path if and only if all its vertices are distinct. We say that P is an **$x - y$ path** if it can be represented by an $x - y$ walk.*

Occasionally, such as when we discuss Euler tours in Chapter 8 or increasing walks in Chapter 16, we will be interested in walks for their own sake. For the most part, though, walks are convenient because they're an easy-to-check version of paths. To know that a sequence of vertices is a walk, we only have to check that consecutive vertices are adjacent. To know that a sequence of vertices represents a path, we have to check a global condition: that none of the vertices in the sequence repeat.

"But wait," you might ask, "Why are you acting like we can choose which definition to check? Surely if a problem calls for a path, we need to find a path, and if a problem calls for a walk, we need to find a walk."

Well, if all we care about is whether one of these objects exists, it turns out that it doesn't matter which one we use! We will prove the following theorem later in this chapter:

Theorem 3.1. *Let x and y be two vertices of a graph G. Then there is an $x - y$ walk in G if and only if there is an $x - y$ path in G.*

Theorem 3.1 means that if we need to prove that one of these objects exists, it's enough to aim for an $x - y$ walk: this will be easier. On the other hand, if one of our assumptions is that one of these objects exists, we can pick either an $x - y$ walk or an $x - y$ path, whichever is convenient.

3.2 Connected graphs

Whether it's in the three cups puzzle, the Towers of Hanoi puzzle, or a complicated graph that appears in serious and dignified study of math, the first question we want an answer for is the existence question: given vertices x and y, does an $x - y$ walk exist in the graph?

In the simplest scenario, the answer is always "yes":

Definition 3.2. *A graph G is **connected** when an $x - y$ walk exists for all $x, y \in V(G)$.*

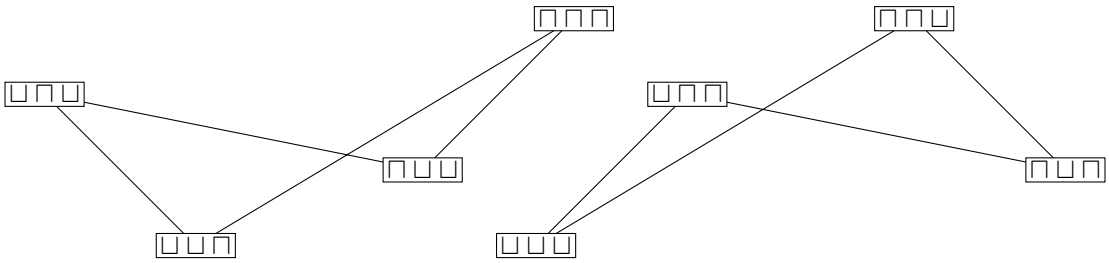


Figure 3.2: The two connected components of the three cup puzzle graph

The state graph of the three cups puzzle is not connected. We can demonstrate this fact (both in this specific example, and in general) by showing how to split the vertex set into two parts with no edges between them. This is done in Figure 3.2, where the two halves are "pulled apart": it is much easier to see here that there are no edges between $\{\sqcup\sqcap\sqcup, \sqcup\sqcup\sqcap, \sqcap\sqcup\sqcup, \sqcap\sqcap\sqcap\}$ and $\{\sqcup\sqcup\sqcup, \sqcup\sqcap\sqcap, \sqcap\sqcap\sqcup, \sqcap\sqcup\sqcap\}$.

When a graph is defined by a sufficiently nice combinatorial rule, such a split often comes with a reason behind it. That is the case with the three cups puzzle. We could in theory confirm that it has no solution by checking each of the states $\sqcup\sqcup\sqcup$, $\sqcup\sqcap\sqcap$, $\sqcap\sqcap\sqcup$, and $\sqcap\sqcup\sqcap$ to verify that all possible moves from these states lead to another one of these states. But we obtain the most insight—and have to exert the least effort—if we notice that the two halves of the graph are characterized by whether the number of $\sqcup$'s (cups which are right side up) is even or odd. A move either increases the number of $\sqcup$'s by two, leaves it the same, or decreases it by two; therefore it's impossible to go from an odd-$\sqcup$ state like $\sqcup\sqcup\sqcup$ to an even-$\sqcup$ state like $\sqcap\sqcap\sqcap$.

Let me describe this splitting tactic more generally, by a lemma we will be able to use in later chapters.

Lemma 3.2. *For a graph G, if there is a set of vertices S such that S is not empty, S is not all of $V(G)$, and there are no edges with exactly one endpoint in S, then G is not connected.*

Proof. Suppose such a set S exists; let $x \in S$ (guaranteed to exist since S is not empty) and let $y \in V(G) - S$ (guaranteed to exist since S is not all of $V(G)$). Then we will show that there is no $x - y$ walk in G.

Assume for contradiction that an $x - y$ walk $(x_0, x_1, \ldots, x_l)$ exists, and let x_i be the first vertex of the walk which is not in S. Then the edge $x_{i-1}x_i$ has exactly one endpoint in S, contradicting our choice of S: $x_i \in S$, but $x_{i-1} \notin S$.

Question:	Why does x_i exist?
Answer:	Because $y \notin S$, so not every vertex of the walk is in S; of the vertices not in S, there must be a first vertex.

Question:	Why does x_{i-1} exist? (And why should we ask that question?)
Answer:	We must ask that question because if $i = 0$, then there is no vertex "x_{-1}" in the walk. This is something we must always watch out for when dealing with subscripts like $i - 1$ or $i + 1$. But in this case, we know $i \neq 0$, because $x_0 = x$ and $x \in S$, but $x_i \notin S$.

We've arrived at a contradiction, so our assumption of an $x-y$ walk must be invalid. Therefore G is not connected. □

Lemma 3.2 is not a very deep result. I want to stop and point it out anyway, because it's important from a proof-writing point of view. You see, when working directly from the definition, "G is not connected" is an awkward statement to prove: we're trying to prove that for some vertices x and y, an $x - y$ walk does not exist, and proving that something does not exist is tricky. The lemma gives us a different target to aim for: to prove that G is not connected, we have to find the set S. Moreover, the properties of S we have to check to apply the lemma are simple ones.

A single set S is enough to show that a graph is not connected, but sometimes it is possible to split the graph into even more pieces with no edges between them. The most fine-grained split possible is a split into connected components: writing the graph G as a disjoint union of any number of connected graphs.

Definition 3.3. *The **connected components** $G_1, G_2, \ldots, G_k$ are subgraphs of G such that*

- *For all i, G_i is connected. In other words, for all $x \in V(G_i)$ and $y \in V(G_i)$, there is an $x - y$ walk in G_i (and therefore in G).*
- *For all $x \in V(G_i)$ and $y \in V(G_j)$ where $i \neq j$, there is no $x - y$ walk in G.*
- *G is the disjoint union of its connected components.*

In order to satisfy the third part of the definition, connected components must be induced subgraphs. We could define a connected component on its own as a subgraph that is connected, but cannot be made any bigger and still remain connected. I did not do this because I think a big part of the definition is that together, the connected components of G tell the entire story of when $x - y$ walks exist in G.

Definition 3.3 is a definition that should be accompanied by a theorem: I've given you no proof of the claim that every graph can be written as the disjoint union of such subgraphs. It is too easy to believe that this is true with no justification, so before we prove it, let me try to persuade you that it isn't quite so obvious after all.

Suppose, for example, that the edges in our graphs stopped being symmetric: that you could walk somewhere, and not be able to get back. In that case, instead of being able to split the graph into connected components, we'd end up with a hierarchy of vertices from which we can reach fewer and fewer destinations—maybe even with some "dead end" vertices that we can walk to, but never leave.

Or suppose that, unlike the infinitely-patient walker that we imagined when defining walks, we consider a walker that gets tired, so that our walks should contain at most (say) 10 steps. Then

it's possible that by going in different directions from vertex x, you can reach both vertices y and z, and yet there is no component containing x, y, and z—because y is too far from z to walk between the two.

So you should take a moment to appreciate the simplicity of the answer we're about to get to the question, "When is there an $x - y$ walk in graph G?" It should be a bit remarkable that the answer will be, "We can split up G into several parts called connected components, and the answer is 'yes' whenever x and y are in the same part, and 'no' otherwise." In a slightly different world, the answer could have been much more complicated!

3.3 Equivalence relations

To understand the connected components of a graph G, we first define a relation $\leftrightsquigarrow$ on pairs of vertices $x, y \in V(G)$: $x \leftrightsquigarrow y$ if there is an $x - y$ walk in G. This relation is often not given any kind of formal name, but it's sometimes called "reachability": $x \leftrightsquigarrow y$ means that from x, we can reach y by following edges in G.

Question:	In the three cup puzzle, for which vertices x does $x \leftrightsquigarrow \sqcup\sqcup\sqcap$ hold?
Answer:	For the vertex $\sqcup\sqcup\sqcap$ itself, for its neighbors $\sqcup\sqcap\sqcup$ and $\sqcap\sqcap\sqcap$, and finally for the vertex $\sqcap\sqcup\sqcup$ that can reach $\sqcup\sqcup\sqcap$ in two steps.

To proceed, we will need to prove the following result:

Lemma 3.3. *The relation $\leftrightsquigarrow$ is an equivalence relation on $V(G)$.*

Before we do that, though, let's talk about equivalence relations.

You may have already seen the definition elsewhere (or not). An *equivalence relation* $\sim$ on a set S is a relation (that is, the statement $x \sim y$ is either true or false for all $x \in S$ and $y \in S$) satisfying three poperties:

1. It is *reflexive*: for all $x \in S$, $x \sim x$.
2. It is *symmetric*: for all $x, y \in S$, if $x \sim y$, then $y \sim x$.
3. It is *transitive*: for all $x, y, z \in S$, if $x \sim y$ and $y \sim z$, then $x \sim z$.

But these are not just nice properties we want to prove because we like how they sound! There is a purpose to showing that something is an equivalence relation. These three properties are exactly the things we need to check in order to know that we can partition S into equivalence classes: non-empty, disjoint sets $S_1, \ldots S_k$ such that $S = S_1 \cup \cdots \cup S_k$ and we have $x \sim y$ exactly when x and y are in the same class.

In the case of our relation $\leftrightsquigarrow$, we will use these equivalence classes to find the connected components of G.

Proof of Lemma 3.3. Let's check all three properties of an equivalence relation.

We check that $\leftrightsquigarrow$ is reflexive: for any vertex x, there is an $x - x$ walk in G. Well, one such walk that's guaranteed to exist is the walk which is a sequence with only one term; (x). There may or may not be others.

We check that $\leftrightsquigarrow$ is symmetric: if there is an $x - y$ walk in G, there is also a $y - x$ walk. Well, suppose that $(u_0, u_1, \ldots, u_l)$ is an $x - y$ walk (with $u_0 = x$ and $u_k = y$). Then reverse it: $(u_l, u_{l-1}, \ldots, u_0)$ is a $y - x$ walk. It starts at y, ends at x, and consecutive vertices in the sequence are adjacent, because they were also consecutive in the $y - x$ walk.

Finally, we check that $\leftrightsquigarrow$ is transitive. Suppose x, y, z are vertices in G such that there is an $x - y$ walk $(u_0, u_1, \ldots, u_l)$ and a $y - z$ walk $(v_0, v_1, \ldots, v_m)$. We know that $x = u_0$, $y = u_l = v_0$, and $z = v_m$. Then there is also an $x - z$ walk: the sequence

$$(u_0, u_1, \ldots, u_l, v_1, v_2, \ldots, v_m).$$

This definitely starts at $x = u_0$ and ends at $z = v_m$. All consecutive vertices are adjacent, because they were already consecutive in the two walks we started with, with the exception of one pair we need to look at more closely: u_l and v_1. These are adjacent because $u_l = y = v_0$, and v_0 and v_1 are consecutive in the $y - z$ walk.

Having checked all three properties, we know that $\leftrightsquigarrow$ is an equivalence relation. □

The proof of Lemma 3.3 gives us a useful operation on walks: we can join an $x - y$ walk and a $y - z$ walk to get an $x - z$ walk.[7] We will make use of this frequently: apart from specifying a walk directly by listing the sequence of vertices, the most common way to construct a walk is by building it out of smaller walks.

Using Lemma 3.3 and the properties of equivalence relations, we know that we can partition $V(G)$ into equivalence classes of $\leftrightsquigarrow$. These equivalence classes are non-empty sets $V_1, V_2, \ldots, V_k$ satisfying three properties:

1. They are pairwise disjoint: if $i \neq j$, then $V_i \cap V_j = \varnothing$.
2. Together, they include all the vertices: $V_1 \cup V_2 \cup \cdots \cup V_k = V(G)$.
3. For $x, y \in V(G)$, we have $x \leftrightsquigarrow y$ if and only if there is a single V_i such that $x \in V_i$ and $y \in V_i$.

These three properties are exactly what we need to say what the connected components of G are, and prove the following theorem:

Theorem 3.4. *Any graph G is the disjoint union of connected components satisfying Definition 3.3.*

[7] Keep in mind that this is not quite the same as joining the sequences together: the last vertex of the $x - y$ walk and the first vertex of the $y - z$ walk are both y, and only one of those y's should be kept in the $x - z$ walk.

Proof. For all i, let G_i be the induced subgraph $G[V_i]$. Property 3 of equivalence classes (as stated above) tells us that an $x-y$ walk exists for two vertices $x, y \in V(G)$ exactly when x and y are both in $V(G_i)$ for some i. That gives us most of the definition of connected components right there.

By the first two properties of equivalence classes, the subgraphs $G_1, G_2, \ldots, G_k$ are disjoint and include every vertex of G. But do they include every edge? Well, for each edge $xy \in E(G)$, the short sequence (x, y) is an $x - y$ walk, so $x \rightsquigarrow y$. This means that there is some V_i such that $x \in V_i$ and $y \in V_i$. But then, because G_i is the induced subgraph $G[V_i]$, we have $xy \in E(G_i)$. We conclude that each edge appears in one of the subgraphs on our list, completing the proof that they are the connected components of G. □

Question:	What are the connected components if G is a connected graph?
Answer:	In this case there is just one connected component: $G_1 = G$. There is a walk from every vertex to every other vertex.

Connected components aren't just useful for answering questions about walks! Because there are no edges between different connected components, they basically do not interact with each other. A lot of the time, if we're asking a question about graphs, we can work with each connected component separately.

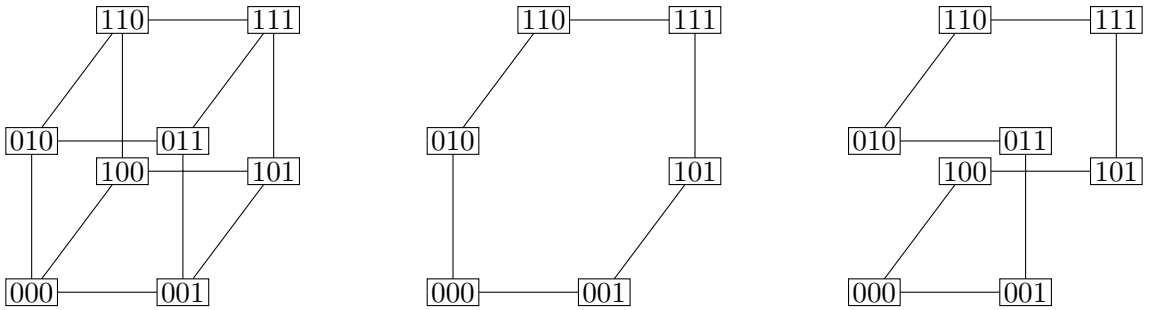
Suppose, for example, that a country's land is separated into two islands, with each island made up of several regions. The two islands will be connected components in a graph that models the country's regions. If we want to find a way to color the map of this country so that adjacent regions get different colors, as we did in the example at the beginning of Chapter 1, then we get separate coloring problems for each island.

Question:	Suppose we want to know if two graphs G and H are isomorphic. How does knowing the connected components of G and of H help?
Answer:	There must be a bijection between their connected components, which pairs each connected component of G with a connected component of H isomorphic to it. For example, suppose G has a 5-vertex connected component and H does not. Then G and H cannot be isomorphic, regardless of what else is going on.

A final consequence of Theorem 3.4 is that when a graph is not connected, we can always use Lemma 3.2 to prove it. If G is the disjoint union $G_1 \cup G_2 \cup \cdots \cup G_k$, where $k > 1$, then $V(G_1)$ is one possibility for a set S satisfying the hypotheses of Lemma 3.2.

3.4 Closed walks and cycles

Once we know where we can start and end a walk, we can start looking at ways that a walk can return to where it started. Adding this condition on its own, though, isn't very useful, because there are many examples that shed absolutely no light on the structure of the graph:



(a) The cube graph Q_3 (b) A 6-cycle in Q_3 (c) A Hamilton cycle in Q_3

Figure 3.3: The cube graph and some cycles in it

- No matter what graph G we take, for any vertex x, the walk (x) ends where it started.
- For every edge $xy \in V(G)$, the walks (x, y, x) and (x, y, x, y, x) and so on give infinitely many more examples.

It is much more interesting to find a cycle in G: a subgraph isomorphic to C_n for some n. Just like paths, cycles can be represented by walks. So we make two definitions: the general one that's easy to check but not useful on its own, and the one that tells us how cycles and walks are related.

Definition 3.4. *A **closed walk** is a walk $(x_0, x_1, \ldots, x_{l-1}, x_0)$. Such a closed walk **represents** a cycle C in G when $V(C) = \{x_0, x_1, \ldots, x_{l-1}\}$ and $E(C) = \{x_0x_1, x_1x_2, \ldots, x_{l-1}x_0\}$.*

The conditions that tell us when a closed walk $(x_0, x_1, \ldots, x_{l-1}, x_0)$ represents some cycle are a bit tricky. First, the vertices $\{x_0, x_1, \ldots, x_{l-1}\}$ all need to be distinct. Second, because a cycle needs at least 3 vertices, we must have $l \geq 3$: the closed walk (x_0) does not represent a cycle.

Question:	If a cycle is represented by the closed walk $(x_0, x_1, \ldots, x_{l-1}, x_0)$, how many closed walks can represent it, in total?
Answer:	There are $2l$ possibilities: we can start at any vertex x_i, and we can go in either direction around the cycle. For example, a cycle represented by (x, y, z, x) could also be represented by (x, z, y, x), (y, x, z, y), (y, z, x, y), (z, x, y, z), and (z, y, x, z).

In Figure 3.1, the closed walk

$$(\sqcup\sqcup\sqcup,\ \sqcup\sqcap\sqcap,\ \sqcap\sqcup\sqcap,\ \sqcap\sqcap\sqcup,\ \sqcup\sqcup\sqcup)$$

represents a cycle, but perhaps at this point in the chapter we need a fresh example.

Suppose that instead of flipping two cups at a time in the three cups puzzle, we could flip any one cup; the puzzle is no longer challenging in any way, but it is more interesting as a mathematical object. So we abandon cups; instead of using the symbols $\sqcup$ and $\sqcap$, we switch to the symbols 0 and 1, so that the vertices are 3-bit sequences. The resulting graph is shown in Figure 3.3a. It is called the *cube graph* because, in addition to the combinatorial description, it

has a geometric one: the corners and (geometric) edges of the unit cube $[0,1]^3$ are exactly the vertices and edges of the cube graph.

We use the notation Q_3 for the cube graph because in Chapter 4, we will generalize this graph to the *hypercube graph* Q_n, with a geometric interpretation in n dimensions.

Question:	In terms of n, how many vertices does Q_n have?
Answer:	2^n vertices: there are 2^n sequences of n bits, because there are 2 choices (0 or 1) for each bit.

The shortest cycles found in Q_3 have 4 vertices, such as the cycle represented by the closed walk $(000, 001, 011, 010, 000)$. We also have 6-vertex cycles such as the one shown in Figure 3.3b: this one is represented by the closed walk $(000, 010, 110, 111, 101, 001, 000)$. There are even cycles through all 8 vertices such as the one represented by $(000, 001, 011, 010, 110, 111, 101, 100, 000)$, which is shown in Figure 3.3c. A cycle through all the vertices of a graph has a special name: it is called a *Hamilton cycle*. We will study these in more detail in Chapter 17.

3.5 Lengths of walks

Finally, once we know that an $x-y$ walk exists, we ask: how many steps does it need to have?

Definition 3.5. *A walk $(x_0, x_1, x_2, \ldots, x_l)$ has **length** l. We also say that a path or cycle has length l if it is represented by a walk of length l.*

This is one of two possible definitions: it counts the edges used by the walk, but we could also have counted the vertices. We choose to count the edges because it makes more sense: the walk that goes nowhere has length 0, and in an application where the walk is a sequence of steps taken (such as in a puzzle), the length of a walk is the number of steps required.

One slightly strange consequence is that while the cycle graph C_n has length n, the path graph P_n has length $n-1$. For this reason, some authors define P_n to be a path with $n+1$ vertices and n edges. In my opinion, this causes more problems than it solves: I am happier if graph families like K_n, C_n, P_n and others are indexed by the number of vertices they have, whenever possible.

The main reason to define the length of a walk is so that we can look for a shortest $x-y$ walk, avoiding walking back and forth inefficiently. Walks that don't revisit any vertices are exactly the ones that correspond to paths, and in fact, that is how we will prove Theorem 3.1: that a graph has an $x-y$ walk if and only if it has an $x-y$ path.

Proof of Theorem 3.1. If a graph G has an $x-y$ path, the walk representing it is an $x-y$ walk, which proves one direction of the theorem.

For the other direction, suppose G has an $x-y$ walk. In that case, let $(u_0, u_1, \ldots, u_l)$, with $u_0 = x$ and $u_l = y$, be an $x-y$ walk of the smallest length possible. We will show that this walk represents a path.

Suppose not, for the sake of contradiction. If the walk fails to represent a path, then the vertices are not all distinct, which means we can pick two positions i and j with $i < j$ such that $u_i = u_j$. But now, consider the sequence

$$(u_0, u_1, \ldots, u_{i-1}, u_i, u_{j+1}, \ldots, u_l)$$

in which we skip vertices $u_{i+1}, u_{i+2}, \ldots, u_j$. This is still an $x - y$ walk! It still starts at x, still ends at y, and every two consecutive vertices are adjacent because they were also adjacent in the original walk. (This is not obvious for the pair $\{u_i, u_{j+1}\}$, but because $u_i = u_j$, it is the same as the pair $\{u_j, u_{j+1}\}$.)

So we've found an $x - y$ walk which has length $l - (j - i)$: strictly less than l. This contradicts our assumption that we took an $x - y$ walk of the smallest length possible. So a shortest $x - y$ walk cannot have repeated vertices: it must represent an $x - y$ path. □

We are now in a position where we can measure distances in a graph, by the lengths of walks connecting them. This is a graph-theoretic distance, not a geometric one: even if there is a way to measure distances between vertices of a graph G geometrically, this might disagree from the graph-theoretic measure.

For example, imagine a graph defined on the streets in a city by placing vertices at points 1 meter apart along every street, and joining vertices along the same street. The length of a walk in this graph tells us the "walking distance": how far we'd have to go along the streets of the city to get from one point to another. This might be very different from distance between two points "as the crow flies", if the points are very close together but without convenient streets between them. It is the "walking distance", and its generalizations to other situations, that we use to define distance between vertices in a graph.

Definition 3.6. *The **distance** between two vertices x and y in a graph G (sometimes written $d_G(x, y)$, or $d(x, y)$ if the graph is clear from context) is the length of the shortest $x - y$ walk, which we now know is also the length of the shortest $x - y$ path.*

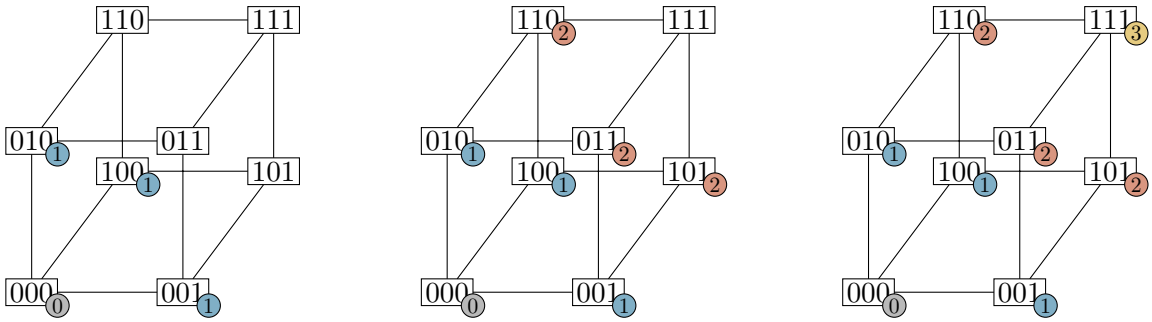
If x and y are in different connected components, then there is no such walk, and we sometimes say that $d(x, y) = \infty$ in that case.

For example, in Figure 3.4c, all vertices of the cube graph Q_3 have been labeled with their distance to the bottom left vertex, 000. Take note of the label 0 on vertex 000 itself. This is there because vertex 000 has distance 0 to itself, as measured by the length-0 walk (000).

What's going on in the other parts of Figure 3.4? These are illustrations of the steps of an algorithm by which all the distances from a single vertex may be computed.

Let x be that starting vertex, in an arbitrary graph G. We begin by knowing only one distance in G: the distance $d(x, x) = 0$.

In Step 1, we take all the neighbors of x; for each vertex y adjacent to x, we set $d(x, y) = 1$. This is the correct distance, because the walk (x, y) has length 1, and there cannot be a length-0 walk from x to any vertex other than x. Moreover, all length-1 walks from x end at a neighbor from x, so we've found all the vertices at distance 1; this will be important later. The end result of Step 1 when $G = Q_3$ is shown in Figure 3.4b.



(a) Step 1 (b) Step 2 (c) Step 3

Figure 3.4: Finding distances in the cube graph

From then on, we repeat the following procedure, of which our Step 1 is a special case. In Step k, we consider every vertex y for which $d(x, y) = k - 1$, and every neighbor z of such a vertex for which $d(x, z)$ is still unknown; we set $d(x, z) = k$. Why is this correct? There's two parts to the verification:

1. First of all, there is an $x - z$ walk of length k. To find it, take the $x - y$ walk of length $k - 1$ (which must exist, assuming the previous steps of our algorithm were correct) and append vertex z to the end of that sequence.
2. Second, there are no shorter $x - z$ walks; that's because, at step $k - 1$, we found all the vertices at distance at most $k - 1$ from x.

Now let's re-confirm point 2 above for Step k, so we can use it in future steps. Suppose that $d(x, z) \leq k$; then for some $j \leq k$, there is a walk $(x_0, x_1, \ldots, x_j)$ with $x_0 = x$ and $x_j = z$. The walk $(x_0, x_1, \ldots, x_{j-1})$ has length $j - 1$, so $d(x, x_{j-1}) \leq j - 1$, and we've found vertex x_{j-1} at Step $j - 1$ or earlier: before Step k. Finally, in the next step after we computed $d(x, x_{j-1})$, we would have considered z (a neighbor of x_{j-1}), and determined $d(x, z)$.

In the cube graph Q_3, the result of Step 2 is shown in Figure 3.4b and the result of Step 3 is shown in Figure 3.4c. Here, if we were to try to do a Step 4, we would accomplish nothing: all neighbors of the vertex labeled with 3 already have known distances. At this point, we stop.

This stopping condition is guaranteed to happen in an n-vertex graph by Step $n - 1$ or earlier: if we were to do more steps than that while computing a new distance at every step, we'd run out of distances to compute. (This is an important thing to check about every algorithm—that it halts!) After the stopping condition is reached, let S be the set of all vertices whose distances to x we've computed. There is a walk between any two vertices in S, because all of them have a walk to x; however, no vertex $y \in S$ has a neighbor $z \notin S$, because we would have computed the value of $d(x, z)$ a step after computing $d(x, y)$ if not earlier.

Therefore $G[S]$ is the connected component of G containing x. If G is connected, then we've computed all distances, and we're done. Otherwise, we can set $d(x, y)$ to ∞ for all $y \notin S$.

Question:	Is there an n-vertex graph for which this algorithm doesn't stop before Step $n-1$?
Answer:	Yes: the path graph P_n, if we start measuring distances from one end of the path graph. In this graph, it takes $n-1$ steps to go from one end to the other.

This distance-finding algorithm is a special instance of breadth-first search, or BFS for short. It is a way of exploring the graph to eventually visit all the vertices in a connected component. It's called "breadth-first" because it tries to go outward from the starting vertex in every direction at once: first a little bit, then a little bit more, and so on.

In the distance-finding algorithm, we used a breadth-first search to compute distances, but this algorithm can be used for other purposes as well; Edward Moore, one of the first people to discover it, used it to find the shortest path through a maze [74]. Later on in the book, we will see other problems that can be solved using a similar algorithm.

3.6 Practice problems

1. Let G be the graph whose vertices are the numbers $1, \ldots, 15$, with an edge between a and b if $|a-b| = 4$ or if $|a-b| = 10$. (For example, vertex 11 is adjacent to vertices 7 and 15, because $|11-7| = |11-15| = 4$, as well as to vertex 1, because $|11-1| = 10$.

   a) Draw a diagram of G.

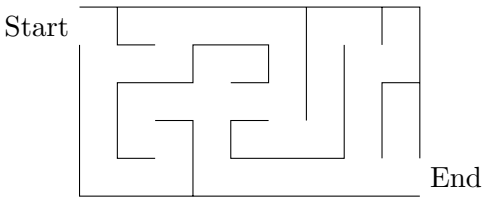
   b) What are the connected components of G?

2. a) Draw a diagram of the circulant graph $\mathrm{Ci}_{10}(1,3)$.

   b) Choose any vertex of $\mathrm{Ci}_{10}(1,3)$, and use the distance-finding algorithm to label every vertex with its distance from your chosen vertex.

   c) The graph $\mathrm{Ci}_{10}(1,3)$ has cycles of four different lengths. What are they? Give an example of each. As a stretch goal, prove that there are no cycles of other lengths; if you do not figure out how, see Chapter 13.

3. How can we find the shortest path through a maze, as Edward Moore did, by using breadth-first search on a graph? Try it out on the maze below:



4. Suppose we generalize the 3-cup puzzle to an n-cup puzzle, where a valid move is to take any two cups that are next to each other and flip them over. Prove that from any starting state with an even number of cups placed right side up, it is possible to reach the state where all n cups are upside down.

5. Prove that the distance between vertices in a connected graph satisfies the following properties, for all vertices x, y, and sometimes z:

   a) $d(x, y) = 0$ if and only if $x = y$.

   b) $d(x, y) = d(y, x)$.

   c) $d(x, z) \leq d(x, y) + d(y, z)$.

   Some of these properties are easier than others, but I am listing them for two reasons. First of all, in some way, they correspond to the proof of Lemma 3.3. Second, they are the three properties that make a connected graph a metric space with metric d; this is not something I will cover in this textbook, but it is interesting in a context outside graph theory.

6. To really understand the proof of Lemma 3.3, it helps to try to generalize it and see what works and what doesn't. Take the following two relations on the vertices of a graph G:

   - $x \frown y$ if there is a $x - y$ walk of odd length.
   - $x \smile y$ if there is a $x - y$ walk of even length.

   For one of these, we can copy the proof of Lemma 3.3 almost word-for-word and prove that it is also an equivalence relation. For the other one, this will not work: there are two places where the proof will fail. Find those places!

7. (Putnam 2010) There are 2010 boxes labeled $B_1, B_2, \ldots, B_{2010}$, and $2010n$ balls have been distributed among them, for some positive integer n. You may redistribute the balls by a sequence of moves, each of which consists of choosing an i and moving exactly i balls from box B_i into any one other box. For which values of n is it possible to reach the distribution with exactly n balls in each box, regardless of the initial distribution of balls?

8. (IMO 1991) Suppose G is a connected graph with k edges. Prove that it is possible to label the edges $1, 2, \ldots, k$ in such a way that at each vertex which belongs to two or more edges, the greatest common divisor of the integers labeling those edges is 1.

4 The degree of a vertex

The purpose of this chapter

After this chapter, I have a whole part of the textbook devoted to the degrees of vertices and the properties of graphs that we can deduce from them. Originally, I had placed this chapter at the beginning of that part, but after writing more of the book, I realized that an introduction to the basics of vertex degrees is too fundamental: the ideas in this chapter will appear over and over again.

To reflect this, I've moved this chapter to be the end of the first part of the textbook. After finishing it, you should feel a bit more free to skip around this book depending on what you're interested in. You may still need to read each part of the textbook in order, and I do assume the knowledge of some things that are covered in previous parts, but there's less interdependence.

The first half of this chapter consists of computational problems that I find fairly light-hearted; the second half contains some more serious proofs, and it may be a spike in difficulty. If you're new to writing proofs in graph theory, it is important to study the proof of Lemma 4.6 until you are comfortable with it; it is a good example of how (and why) to induct on the number of vertices in a graph.

4.1 The hypercube graphs

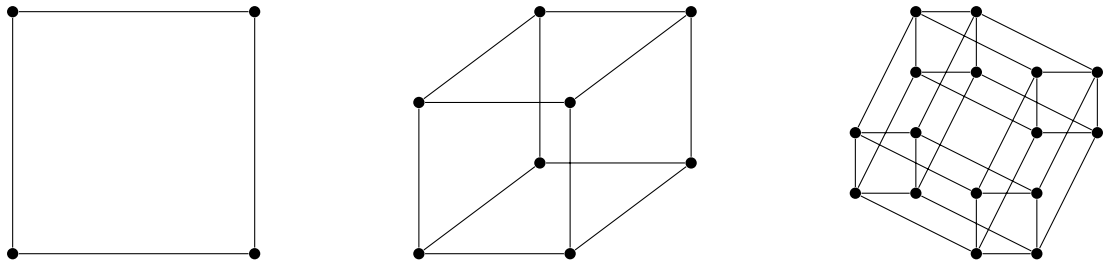
Before we begin discussing vertex degrees, let me introduce a new family of graphs to you: the hypercube graphs. The family of hypercube graphs generalizes the cube graph previously mentioned in Chapter 3:

Definition 4.1. *The **n-dimensional hypercube graph Q_n** has:*

- *Vertex set $V(Q_n) = \{0,1\}^n$, which we will denote with n-bit strings $b_1 b_2 \dots b_n$. For example, $V(Q_2) = \{00, 01, 10, 11\}$.*
- *An edge between every pair of vertices that differ only in one position. For example, in Q_3, vertex 010 is adjacent to vertices 110, 000, and 011.*

*The 3-dimensional hypercube graph Q_3 is also known as the **cube graph**.*

Question:	In Q_4, what are the neighbors of vertex 0101?
Answer:	Vertices 1101, 0001, 0111, and 0100: we get 1101 by flipping the first bit, 0001 by flipping the second bit, 0111 by flipping the third bit, and 0100 by flipping the fourth bit.



(a) Q_2 (b) Q_3 (c) Q_4

Figure 4.1: Three hypercube graphs of three different dimensions

In Appendix B, I give an alternate, recursive definition of Q_n: useful when proving any of its properties by induction on n.

Figure 4.1 shows three examples of hypercube graphs: Q_2, Q_3, and Q_4. You may notice that Q_2 resembles a 2-dimensional square, and Q_3 resembles a 3-dimensional cube; in general, Q_n is the graph that describes the geometrical structure of an n-dimensional hypercube. However, Q_n also has a variety of applications that have nothing to do with geometry:

- For computer scientists, it is the graph of n-bit sequences with adjacency defined by flipping bits. When is that kind of adjacency useful? For example, when designing error-correcting codes, which need to detect accidental bit flips. The code words in an error correcting code are vertices of Q_n all at a high distance from each other.
- We can also think of the vertices of Q_n as subsets of the set $\{1, 2, \ldots, n\}$, where an n-bit string $b_1 b_2 \ldots b_n$ corresponds to the subset $B = \{i : b_i = 1\}$. With this interpretation, the graph theory behind Q_n can tell us about the combinatorics of set families.
- In graph theory, Q_n can be useful as a simple construction of a graph with many vertices and lots of symmetry; additionally, despite having relatively few edges for how many vertices it has, none of the vertices are very far apart.

But "relatively few edges for how many vertices it has" is a vague statement. How many vertices does Q_n have, and how many edges?

To count the vertices, it's enough to think about the number of ways to choose an element $b_1 b_2 \ldots b_n \in Q_n$: for each bit b_i, there are 2 choices, and each choice can be made independently of the others, so there are 2^n choices overall.

To find the number of edges in Q_n, we will first develop a bit of the theory of vertex degrees, which will make the problem much easier.

4.2 The handshake lemma

Let's begin with the main definition:

Definition 4.2. *The **degree** of a vertex is the number of incident edges it has. The degree of vertex x in a graph G is written $\deg_G(x)$, or just $\deg(x)$ if the graph is clear from context.*

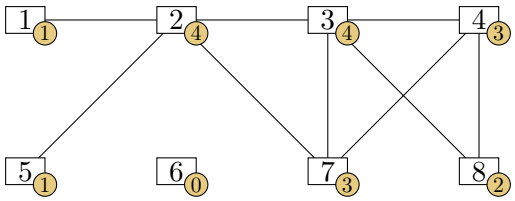


Figure 4.2: An 8-vertex graph labeled with the degrees of its vertices

Figure 4.2 shows an 8-vertex graph with vertices labeled according to their degree.

Question:	In a graph with 8 vertices, what is the largest number that could be the degree of a vertex?
Answer:	7: if a vertex x is adjacent to every other vertex, then $\deg(x) = 7$. In general, in an n-vertex graph, the largest degree possible is $n - 1$.

There is a lot of associated terminology.

Definition 4.3. *An **isolated vertex** is a vertex whose degree is* 0*; a **leaf** is a vertex whose degree is* 1*.*

Isolated vertices are interesting because they are the smallest possible connected components in a graph. Leaves are less obviously notable; we will (appropriately) find out what's interesting about them when we study trees in Chapter 10.

Definition 4.4. *The **maximum degree** $\Delta(G)$ of a graph G is the largest degree of any vertex. The **minimum degree** $\delta(G)$ of a graph G is the smallest degree of any vertex.*

The notation $\Delta(G)$ and $\delta(G)$ is only the beginning of a general trend to use Greek letters for numerical properties of graphs; for the important properties, these letters are mostly standard. (If you are unfamiliar with the Greek alphabet, Δ and δ are an uppercase and lowercase "delta", respectively; this is a "d" for "degree", uppercase for maximum and lowercase for minimum. It's not entirely random.)

The handshake lemma, or degree sum formula, is the first tool that gives degrees any sort of purpose. In my opinion, it is one of the main results from graph theory that mathematicians from all fields should know: not just for its statement, but to use as a way of thinking about graphs. The other result that I'd put in that category is Hall's theorem (Theorem 15.1), which we'll prove in Chapter 15.

Lemma 4.1 (Handshake lemma)**.** *In any graph G, the vertex degrees add up to twice the number of edges:*

$$\sum_{v \in V(G)} \deg_G(v) = 2|E(G)|.$$

Proof. Many proofs exist; for the sake of practice, let's do a proof by induction. We will prove that for any graph with m edges, the sum of degrees is $2m$, by induction on m.

The base case is $m = 0$. Here, we have a graph with no edges. No matter how many vertices we have, their degrees are all 0, the sum of the degrees is 0, and $2m$ is also 0.

Assume that the degree sum formula holds for all $(m-1)$-edge graphs. Let G be a graph with $m \geq 1$ edges, and let xy be any edge of G. We can apply the induction hypothesis to $G - xy$ (the graph we get by deleting edge xy from G), a graph with $m - 1$ edges.

What is the relationship between the degrees of G and the degrees of $G - xy$? Both x and y have one extra incident edge in G that they don't have in $G - xy$: edge xy itself. So

$$\deg_G(x) = 1 + \deg_{G-xy}(x) \text{ and } \deg_G(y) = 1 + \deg_{G-xy}(y).$$

For any other vertex v, $G - xy$ and G have the same number of edges, so we have

$$\deg_G(v) = \deg_{G-xy}(v).$$

Also, $G - xy$ and G have the same set of vertices. So if we add up the vertex degrees in $G - xy$ and G, the result is that

$$\sum_{v \in V(G)} \deg_G(v) = 2 + \sum_{v \in V(G-xy)} \deg_{G-xy}(v).$$

Applying the induction hypothesis, we get that the degree sum in $G - xy$ is $2(m-1)$, so the degree sum in G is $2(m-1) + 2 = 2m$.

By induction, the degree sum formula holds for all graphs. □

Using the handshake lemma, we can immediately answer our earlier question: how many edges does the hypercube Q_n have? There are 2^n vertices in Q_n, and each of them has n neighbors: every bit sequence $b_1 b_2 \dots b_n$ has n places in which a bit an be toggled. So the sum of degrees is $n \cdot 2^n$, and therefore the number of edges is $n \cdot 2^{n-1}$.

Here are some other questions we can answer quickly using the handshake lemma:

Problem 4.1. *In particular, Q_3 has 8 vertices of degree 3. Can we have a 7-vertex graph where all the vertices have degree 3?*

Answer to Problem 4.1. No: then the degree sum would be $7 \cdot 3 = 21$, so there would be 10.5 edges, which is impossible. □

Problem 4.2. *A soccer ball has 12 black pentagonal panels (and some white hexagonal panels I'm too lazy to count). Panels are stitched along their edges, and meet at corners; at each corner, a pentagon and two hexagons meet. How many edges are there where two panels meet?*

Answer to Problem 4.2. Each black pentagon has 5 corners, which will be the $12 \cdot 5 = 60$ vertices of our graph; the edges will be the edges where panels meet. Here, each vertex has degree 3, so the sum of degrees is $60 \cdot 3 = 180$, and there are 90 edges.

(Some slightly fancier logic can convince us that there are 20 hexagons; see the practice problems at the end of this chapter.) □

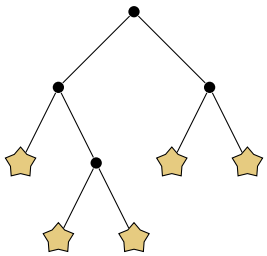


Figure 4.3: An full binary tree with 8 vertices; the marked vertices are leaves.

Problem 4.3. *Suppose you have a graph G with 9 vertices and 20 edges. What can the minimum degree $\delta(G)$ of this graph be?*

Answer to Problem 4.3. Since the sum of the degrees is $2 \cdot 20 = 40$, the average of the degrees is $\frac{40}{9} \approx 4.44$. So the minimum degree can be at most 4: if every vertex had degree 5 or more, then the sum of degrees would be at least $5 \cdot 9 = 45$. We can find 9-vertex, 20-edge graphs with a minimum degree of each of 0, 1, 2, 3, or 4; try it yourself! □

Problem 4.4. *In computer science, a full binary tree is a graph where every vertex is adjacent to up to 3 vertices: a parent and up to 2 children. Every vertex except one (the root) has a parent; every vertex has either 0 children or 2 children. (The parent-child relationship is symmetric.) Figure 4.3 shows a 9-vertex full binary tree; 5 of its vertices are leaves. In a 99-vertex full binary tree, how many leaves would there be?*

Answer to Problem 4.4. Suppose there are k leaves, which each have degree 1; there is 1 root, with degree 2, leaving $99 - k - 1$ vertices with degree 3 (a parent and 2 children). So the total degree is $k + 2 + 3(99 - k - 1)$ or $3 \cdot 99 - 2k - 1$.

By the handshake lemma, this is twice the number of edges. But we can count the edges differently! Each edge is between a parent and a child; all 98 vertices except the root have a parent, so there are 98 edges. Setting $3 \cdot 99 - 2k - 1$ equal to $2 \cdot 98$ and solving for k, we get $k = 50$: there must be 50 leaves. □

We can generalize the answer to Problem 4.1:

Corollary 4.2. *Every graph G must have an even number of vertices (possibly 0) whose degree is odd.*

Proof. What is the parity of the sum of degrees of G: is it odd or even?

One way to answer that question is simply to add up all the degrees. Starting from 0 (an even number), when we add an even degree to the total, it does not change the parity; when we add an odd degree to the total, it flips the parity. So if the number of odd degrees is even, the parity is flipped an even number of times, and ends at an even number; if the number of odd degrees is odd, the parity is flipped an odd number of times, and ends at an odd number.

Another way to answer the question, however, is to use the handshake lemma. The sum of degrees of G is definitely an even number, because it's twice the number of edges!

For the two answers to agree, as we know they must, there must be an even number of vertices with odd degree. □

4.3 Degrees and connectedness

Here's another way we can use the minimum degree of a graph:

Proposition 4.3. *Let G be a graph with n vertices whose minimum degree $\delta(G)$ is at least $\frac{n-1}{2}$. Then G is connected.*

Proof. We want to show that given any two vertices x and y, G must have an $x-y$ walk.

This is definitely true if xy is an edge of G: in this case, the sequence (x, y) is an $x-y$ walk. So let's suppose that $xy \notin E(G)$.

There are $n-2$ vertices in G other than x and y. Of these $n-2$ vertices, at least $\frac{n-1}{2}$ are adjacent to x, and at least $\frac{n-1}{2}$ are adjacent to y. Together, $\frac{n-1}{2} + \frac{n-1}{2}$ adds up to more than $n-2$, so there must be some overlap! That overlap is a vertex z adjacent to both x and y.

In this case, (x, z, y) is an $x-y$ walk, completing our proof. □

(Actually, we've shown more: we've shown that the distance between any two vertices is at most 2. In such cases, we say that G has *diameter* at most 2: more on this topic in Chapter 5.)

Proposition 4.3 is a very common type of theorem to prove; it is, in some sense, our first glimpse of extremal graph theory. Extremal graph theory is the sub-discipline of graph theory that explores the relationships between possible values of different graph properties. Here are a couple of ways this can go:

- Suppose we consider two numerical invariants of graphs: $x(G)$ and $y(G)$. It is very rare that knowing $x(G)$ will tell us $y(G)$, if we've bothered to define both numbers at all. However, maybe knowing $x(G)$ will give us a range of possible values of $y(G)$: there is a region in the xy-plane where the points $(x(G), y(G))$ can go.

  To describe that region, it's natural to explore its boundaries. That's why extremal graph theory is "extremal": it studies the extreme (highest or lowest) values that one invariant can have, based on another. We might describe the relationship between the two invariants by an inequality: for example, the inequality $\delta(G) \leq \Delta(G)$ is a rather silly statement of extremal graph theory.

- Suppose we consider a numerical invariant $x(G)$ and a property P that a graph G either has or does not have. Then the "extremal" question we can ask is this: among graphs that have P, what is the range of possible values of $x(G)$? What about graphs that do not have P? Using this, we might be able to deduce whether a graph G has property P or not, based on the value of $x(G)$.

Proposition 4.3 is an example of the second type of statement, describing the relationship between minimum degree and connectedness. It is very common in extremal graph theory to ask: what is a lower bound $\delta(G) \geq ___$ that will guarantee that G has a certain property?

Of course, we have to try to find a good lower bound. It is much easier to prove the statement, "If an n-vertex graph G has minimum degree $n-1$, then G is connected," because the only n-vertex graph G with minimum degree $n-1$ is the complete graph K_n. However, Proposition 4.3 is stronger: it applies to more graphs. Is it as strong as possible?

To answer that question, we look for a so-called "extremal example". If we can find a graph G which just barely fails the condition $\delta(G) \geq \frac{n-1}{2}$, but is not connected, then we know that Proposition 4.3 cannot be improved any further: that graph G is the limit. (Ideally, we'd find such a graph for many possible values of n, to give an example that applies to all situations.)

There is such an example. Suppose n is even, and G has two connected components which are complete graphs with $\frac{n}{2}$ vertices each. Then each vertex is adjacent to the $\frac{n}{2}-1$ other vertices in its component, and $\delta(G) = \frac{n}{2} - 1$, yet G is not connected.

Question:	Suppose that n is odd: $n = 2k+1$ for some k. What is the largest possible degree in a graph G which is not connected?
Answer:	It is $k-1$, which we can achieve when one component of G is a k-vertex complete graph, and the other is a $(k+1)$-vertex complete graph.

Question:	How do we know we can't do better than this example?
Answer:	The next integer after $k-1$ is k, which is exactly equal to $\frac{n-1}{2}$: at this point, by Proposition 4.3, we know that the graph must be connected.

4.4 Degrees and cycles

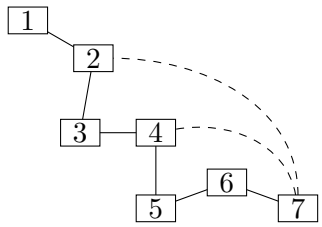
The following theorem, and Corollary 4.7 in the next section, will both be very useful to us many times in future chapters; notably, in Chapter 8 to find cycle decompositions and in Chapter 10 to study graphs with no cycles. In the meantime, they will show a few other ways we can use vertex degrees in proofs.

Theorem 4.4. *Every graph G whose minimum degree $\delta(G)$ is at least 2 contains a cycle.*

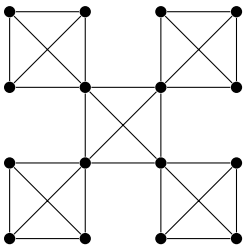
Proof. Let the walk $(x_0, x_1, \ldots, x_l)$ represent a longest path in G.

We know $\delta(G) \geq 2$ and therefore in particular $\deg(x_l) \geq 2$. Is it possible that x_l has a neighbor y which is not one of $x_0, x_1, \ldots, x_{l-1}$? No! In that case, the walk $(x_0, x_1, \ldots, x_l, y)$ would represent a longer path.

So x_l has at least two neighbors, and they are all in the set $\{x_0, x_1, \ldots, x_{l-1}\}$. Figure 4.4a shows an example in which the walk representing the longest path is $(1, 2, 3, 4, 5, 6, 7)$, and $x_l = 7$ is adjacent to vertices 2, 4, and 6.



(a) From a longest path to a cycle



(b) An extremal example

Figure 4.4: Illustrations for Theorem 4.4 and Proposition 4.5

One of x_l's neighbors is x_{l-1}: the previous vertex on the path. This doesn't help us. But there must be another vertex x_i with $i < l-1$ which is also adjacent to x_l.

Then the closed walk $(x_i, x_{i+1}, \ldots, x_l, x_i)$ represents the cycle we wanted. □

Question:	How do we know that a longest path in G exists?
Answer:	There's an upper limit to the length of a path: in an n-vertex graph, there can be no path of length n or larger, because such a path would need to contain $n+1$ different vertices.

Question:	Why does this matter for our proof?
Answer:	If we begin our proof by taking a longest path in G, then it had better be the case that such an object exists. We could not begin a proof by saying, "Let $(x_0, x_1, x_2, \ldots, x_l)$ be a longest walk in G," because there are often arbitrarily long walks.

What if we raise the minimum degree, and assume $\delta(G) \geq 3$, or $\delta(G) \geq 10$? In that case, we have greater flexibility in choosing vertex x_i, because x_l will have more neighbors among $\{x_0, x_1, \ldots, x_{l-2}\}$. One possible way to use that flexibility is to strengthen the result in Theorem 4.4 by choosing the vertex x_i that will give us the longest cycle. For example, in Figure 4.4a, going from vertex 7 back to vertex 4 gives us a cycle of length 4, but going back all the way to vertex 2 gives a cycle of length 6.

What lower bound can we prove on the length of this cycle, in terms of $\delta(G)$? This is an instance in which, if you are feeling sufficiently motivated, you should stop reading and take a moment to try to state, and prove, a stronger result on your own.

If not, join me and continue!

Proposition 4.5. *Every graph G with $\delta(G) \geq 2$ contains a cycle of length at least $\delta(G)+1$.*

Proof. As in the proof of Theorem 4.4, if the walk $(x_0, x_1, \ldots, x_l)$ represents a longest path in G, then x_l has at least $\delta(G)$ neighbors, which are all in the set $\{x_0, x_1, \ldots, x_{l-1}\}$.

We are still going to continue by choosing a vertex x_i adjacent to x_l and looking at the closed walk $(x_i, x_{i+1}, \ldots, x_l, x_i)$, but we're trying to find a long cycle.

Question:	Which x_i should we pick to make the closed walk as long as possible?
Answer:	The x_i with the least value of i.

As long as $\delta(G) \geq 2$, the x_i with the least i will come before x_{l-1}, so the closed walk $(x_i, x_{i+1}, \ldots, x_l, x_i)$ really will represent a cycle. (All that's necessary here is that it has length at least 3.) What's more, the vertices of the cycle include x_l as well as every neighbor of x_l, because we went back to the earliest neighbor we could. This tells us that the cycle has at least $\delta(G) + 1$ vertices, and therefore its length is at least $\delta(G) + 1$. □

As before, we should ask if this is the best possible result: can we find examples where no cycle has length more than $\delta(G)+1$? One example where we cannot exceed this is a complete graph: in K_n, the minimum degree is $n-1$ and no cycle has length more than n.

Figure 4.4b shows a more satisfying kind of example, in my opinion. Here, we take the union of several copies of K_n that overlap in individual vertices. If we arrange them with some care, then the minimum degree is still $n-1$ and there is still no cycle of length more than n. But the number of vertices can be made arbitrarily large, so it is clear that it's not the limiting factor.

4.5 Average degree

Even if you have lots and lots and lots of edges, your minimum degree can be very small. For example, a 100-vertex graph might consist of a 99-vertex complete graph and a single isolated vertex. This has 4851 edges, which is close to the maximum number of edges a 100-vertex graph can have: 4950. And yet the minimum degree is 0, and we can't apply any nice results like Theorem 4.4 to this graph.

The following lemma partially solves this problem, and is very commonly used in extremal graph theory. It was first proved by Pál Erdős in 1964 [28] to solve a problem relating average degree to the existence of certain subgraphs, though special cases of it were used even earlier.

Lemma 4.6. *Let G be a graph with average degree at least d, where d is a positive integer. Then G contains a subgraph H with $\delta(H) > \frac{d}{2}$.*

Proof. We induct on n, the number of vertices in G. (This proof carefully follows the induction template from Appendix B, to avoid any errors.)

When $n \leq d$, the theorem "holds trivially": that is, when $n \leq d$, the highest possible degree in G is $d-1$, so G cannot have average degree d or more to begin with! We could use that as our base case, but it's a bit more satisfying to use $n = d+1$.

In this case, the highest possible degree in the graph is d. The average degree can only be this high if every vertex has degree d: if $G = K_{d+1}$. In this case, G itself is the subgraph H we're looking for. This base case also holds.

Either way, suppose that the theorem holds for all $(n-1)$-vertex graphs with average degree at least d. Let G be an n-vertex graph with average degree at least d.

At this point, let's say something about average degree. If G has vertices $x_1, x_2, \ldots, x_n$, then the average of the degrees is

$$\frac{\deg(x_1) + \deg(x_2) + \cdots + \deg(x_n)}{n} = \frac{2|E(G)|}{n},$$

by the handshake lemma. This equation tells us that the average degree of G is $\frac{1}{2}n$ times the number of edges in G. In particular, the statement "G has average degree at least d" is equivalent to the statement "G has at least $\frac{1}{2}nd$ edges".

We're assuming G has average degree at least d, so we're assuming that G has at least $\frac{1}{2}nd$ edges. Also, if $\delta(G) > \frac{d}{2}$, then we are already done: we were looking for a subgraph with this minimum degree, and G itself can be that subgraph! So we can add on a further assumption for free: that G has a vertex x with $\deg(x) \leq \frac{d}{2}$.

In that case, let $G' = G - x$: the graph obtained from G by deleting x. We know that G' has $n-1$ vertices and at least $\frac{1}{2}nd - \frac{d}{2}$ edges: we started with at least $\frac{1}{2}nd$ edges, and we lost at most $\frac{d}{2}$ of them from deleting x. This simplifies to $\frac{1}{2}(n-1)d$, so G' has at least $\frac{1}{2}(n-1)d$ edges, which means G' has average degree at least d, too!

By applying the induction hypothesis to G', we learn that G' has a subgraph H with $\delta(H) > \frac{d}{2}$. Since H is a subgraph of G', and G' is a subgraph of G, we have found the subgraph of G we wanted.

By induction, the theorem holds for graphs with any number of vertices. □

Using Lemma 4.6, we can convert minimum-degree results to average-degree results. For example, we can use Lemma 4.6 to turn Theorem 4.4 into a result about average degree—or, equivalently, about the number of edges.

Corollary 4.7. *If G has n vertices and at least n edges, then G contains a cycle.*

Proof. If G has n vertices and at least n edges, it has average degree at least $d = 2$. By Lemma 4.6, G has a subgraph H with $\delta(H) > \frac{d}{2} = 1$. If $\delta(H) > 1$, then $\delta(H) \geq 2$, so by Theorem 4.4, H contains a cycle. This is also a cycle in G. □

Question:	Is the lower bound "at least n edges" in Corollary 4.7 the best lower bound possible?
Answer:	Yes: a graph with n vertices and $n-1$ edges does not need to contain a cycle. For example, the graph P_n has n vertices, $n-1$ edges, and no cycles.

4.6 Practice problems

1. Suppose that G is a graph with 5 vertices and 7 edges. For which pairs (a, b) is it possible that $\delta(G) = a$ and $\Delta(G) = b$?

   For the cases where it is possible, give an example. For the cases where it is not possible, explain why not.

2. The five *Platonic solids* (defined in more detail in Chapter 23) are:

   - The tetrahedron, which has 4 vertices and 4 triangular faces, with 3 faces meeting at every corner.
   - The cube, which has 8 vertices and 6 square faces, with 3 faces meeting at every corner.
   - The octahedron, which has 6 vertices and 8 triangular faces, with 4 faces meeting at every corner.
   - The dodecahedron, which has 20 vertices and 12 pentagonal faces, with 3 faces meeting at every corner.
   - The icosahedron, which has 12 vertices and 20 triangular faces, with 5 faces meeting at every corner.

   In each case, find the number of edges where two faces meet.

3. Let's return to the soccer ball in Problem 4.2. Let the soccer ball have 12 pentagons and x hexagons. Each pentagon borders 5 hexagons; each hexagon borders 3 pentagons and 3 other hexagons.

   Consider a different graph describing the soccer ball: its vertices are the pentagonal and hexagonal panels, and two vertices are adjacent whenever the panels are stitched together.

   a) Use the handshake lemma to compute the number of edges in this graph.

   b) Use the handshake lemma on a subgraph of this graph to find the number of edges corresponding to hexagon-to-hexagon boundaries.

   c) Solve for x.

4. We deduced Corollary 4.7 on the existence of cycles by combining Theorem 4.4 and Lemma 4.6.

   In a similar way, we can combine Proposition 4.3 on connectedness and Lemma 4.6 to find an average degree (or, equivalently, a minimum number of edges) which will guarantee guarantee that a graph is connected.

   a) What minimum number of edges do you get in this way?

   b) Is it the best possible statement of its type? Try to construct an n-vertex graph with as many edges as possible that is not connected.

c) Without using Lemma 4.6, prove that when $n \geq 3$, every n-vertex graph with at least $\binom{n}{2} - (n-2)$ edges is connected.

(You should think of this bound in the following way: every n-vertex graph which is missing at most $n-2$ edges is connected.)

5. Let G_n be the subgraph of Q_n induced by the vertices in which the total number of 1's is either 1 or 2. (For example, the vertices of G_3 are $\{001, 010, 100, 011, 101, 110\}$.)

   a) Draw a diagram of G_4, and find the degree of each vertex.

   b) How many edges does G_n have, as a function of n?

6. a) A connected graph G has the property that for every edge xy, $\deg(x) = \deg(y)$. Prove that all vertices of G have the same degree.

   b) A connected graph G has the property that for every vertex x, the degree of x is at most the average degree of the neighbors of x. That is, if the neighbors of x are $y_1, y_2, \ldots, y_k$, then

$$\deg(x) \leq \frac{\deg(y_1) + \deg(y_2) + \cdots + \deg(y_k)}{k}.$$

   Prove that all vertices of G have the same degree.

7. An open problem called Conway's 99-graph problem is to determine whether there is a 99-vertex graph with the following properties:

   - Every two adjacent vertices have exactly one common neighbor;
   - Every two non-adjacent vertices have exactly two common neighbors.

   If such a graph exists, then every vertex in it must have the same degree, d. What is d?

   (I am not asking you to find the graph. John Conway offered a $1000 reward [19] for a solution to the problem! That gives you an idea of how hard that would be.)

8. (IMO 1971) Prove that for every positive integer m we can find a finite set S of points in the plane, such that given any point A of S, there are exactly m points in S at unit distance from A.

Vertex Degrees

5 Regular graphs

The purpose of this chapter

Most chapters in this book are all about giving you tools to deal with any graph you encounter. This chapter has a bit of that, but it is much more about showing you examples of some particular useful graphs.

In Theorem 5.1, we use circulant graphs (which we previously defined in Chapter 2) to construct examples of regular graphs in every case where it is possible. In the next section, I attempt to convey some of the variety of possible regular graphs out there (except, of course, in the cases without much variety). Next, we meet the Petersen graph, which will appear as an example or counterexample many times throughout this book: it will be especially useful to have in our back pocket when we get to Chapter 17 and Chapter 20.

The last section discusses the degree/diameter problem; this problem is not exclusive to regular graphs, but many of the examples found when making progress on it are regular. It is also an example of a research problem in graph theory where the main form of progress on it is in fact by finding examples.

There is no general way to find examples; otherwise, problems like the degree/diameter problem would already be solved. However, being familiar with a wide variety of examples, and knowing what they're good for, is valuable when it comes to writing existence proofs. The proof of Theorem 5.1 would have been much harder if we did not already have circulant graphs at our disposal.

In this chapter, we take our first serious look at necessary and sufficient conditions. This is a theme we will return to throughout the book; if you need an introduction to necessary and sufficient conditions, you can find it in Appendix A.

5.1 Degree sequences

In Chapter 4, we looked at the degrees of individual vertices; now, let's look at all of them together.

Definition 5.1. *The **degree sequence** of a graph G is a sequence of numbers that gives all the degrees of all the vertices of G.*

Despite the word "sequence", these don't come in any particular order, because the vertices of a graph don't have a particular order. (There might be a natural order you're tempted to use based on the way we drew a graph, but we don't want the properties of graphs to depend on how we draw them!) It would have been more reasonable to talk about the "degree multiset"

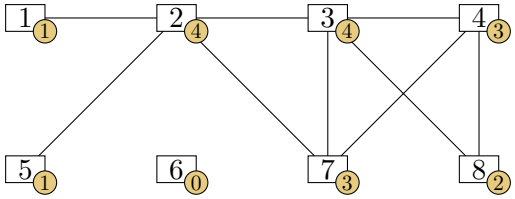


Figure 5.1: An 8-vertex graph labeled with the degrees of its vertices

of a graph: a multiset keeps track of how often its elements appear, but not their order, which is exactly what we want here.

Instead, the convention is to write the degree sequence of a graph in descending order.[8] We do this to avoid the temptation of looking for meaning in the order of the numbers, and because it will be convenient for working with the degree sequence. For example, we would say that Figure 5.1 shows a graph with degree sequence $4, 4, 3, 3, 2, 1, 1, 0$.

It is a relatively quick and painless process to examine a graph, count the number of lines poking out of each dot, and in doing so compute the degree sequence. For this reason, we often look to the degree sequence first to try to learn about a graph; even if it does not always tell us what we want to know, it is a quick way to start! This is, for example, a good way to begin determining whether two graphs are isomorphic.

Question:	If you want to know whether two graphs are isomorphic, what do their degree sequences tell you?
Answer:	If G and H have different degree sequences, they're definitely not isomorphic. If they have the same degree sequences, they might be isomorphic, but it's too early to say. In other words, G and H having the same degree sequence is a necessary condition, but not a sufficient condition for them to be isomorphic.

Though it is easy to compute the degree sequence of a given graph, it is harder to go in reverse: to take a sequence of nonnegative integers in descending order, and determine if it is the degree sequence of some graph. If it is, we call it a **graphic sequence**. For example, $4, 4, 3, 3, 2, 1, 1, 0$ is a graphic sequence, and the graph in Figure 5.1 is a proof of this. (There are other graphs with the same degree sequence, too.)

Here are a few quick puzzles about graphic sequences that illustrate some things you already know about them, and some things you have yet to learn.

Problem 5.1. *Is the sequence* $8, 8, 8, 8, 8, 8, 8, 8$ *graphic?*

[8] I want to make it clear that in this book, sequences in "descending order" or "ascending order" can have ties: consecutive terms may be equal. Sometimes a sequence in descending order is called "non-increasing" to make it clear that ties are allowed, but I think this is awkward.

Answer to Problem 5.1. No: the terms are too big. This sequence would like to be the degree sequence of an 8-vertex graph, but the largest possible degree in an 8-vertex graph is 7, achieved when a vertex is adjacent to all 7 other vertices. □

Problem 5.2. *Is the sequence* $5, 5, 5, 3, 3, 3, 1, 1, 1$ *graphic?*

Answer to Problem 5.2. No: this sequence violates Corollary 4.2 to the handshake lemma, which states that a graph cannot have an odd number of vertices with an odd degree. □

Problem 5.3. *Is the sequence* $4, 3, 2, 1, 0$ *graphic?*

Answer to Problem 5.3. No: suppose for contradiction that G is a graph with degree sequence $4, 3, 2, 1, 0$. Let x be the vertex with degree 4 and let y be the vertex with degree 0. Then x is adjacent to all 4 other vertices, so in particular $xy \in E(G)$; however, y is adjacent to 0 other vertices, so in particular $xy \notin E(G)$. Therefore it is impossible for G to exist. □

Question:	How do the solutions to Problem 5.1 and Problem 5.2 generalize?
Answer:	They give us two rules: in an n-term graphic sequence, the values must be integers between 0 and $n-1$, and an even number of them must be odd.

Question:	Are these rules always enough to tell if a sequence is graphic?
Answer:	No, and that's why Problem 5.3 is there. It's a 5-term sequence in which the values are integers between 0 and 4, and 2 of them are odd; however, the sequence is not graphic.

We should think of the rules we deduced from the solutions to Problem 5.1 and Problem 5.2 as simple preliminary tests we can carry out before thinking about a sequence very hard. They are necessary conditions: if a sequence fails one of the two preliminary tests, it's definitely not graphic. This can save us a lot of work when faced with a larger problem of this type.

However, the preliminary tests are not sufficient conditions: if a sequence passes both preliminary tests, it still might not be graphic. We will eventually develop a comprehensive theory of graphic sequences, which will encompass the argument we used to solve Problem 5.3 as a special case.

5.2 Regular graphs

As a special case, we can consider the graphic sequence problem for a sequence in which all terms are equal:

Definition 5.2. *A* ***regular graph*** *is a graph in which every vertex has the same degree. More specifically, an* ***r-regular graph*** *is a graph in which every vertex has degree r. The degree sequence of such a graph is $r, r, \ldots, r$.*

Making a definition does not guarantee that any object satisfying the definition exists. So when do regular graphs exist?

Question:	Suppose the sequence $r, r, \ldots, r$ with n terms is graphic. What do our two simple preliminary tests tell us about the relationship between r and n?
Answer:	We must have $0 \leq r \leq n-1$, and if r is odd, then n must be even.

In fact, the existence problem for regular graphs is much easier than the general case. There are no further complications, and these are the only two conditions.

Theorem 5.1. *An r-regular graph on n vertices exists whenever $0 \leq r \leq n-1$ and at least one of r or n is even.*

Proof. To prove that an r-regular graph on n vertices exists, we need to construct one.

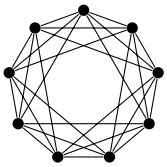
If we were solving the problem for specific values of r and n, the proof would be very short: we could simply draw a diagram and verify that the graph shown has the right number of vertices, all with the correct degree. To solve the problem in general, we need to give a family of graphs as the solution.

Not every family of graphs is sufficiently flexible for our purposes. For example, in Chapter 4, we defined the family of hypercube graphs. These are all regular graphs: for any r, if we want an r-regular graph, the r-dimensional hypercube graph Q_r will do. However, there is only one r-dimensional hypercube graph, and it has 2^r vertices. If we want a regular graph with some other number of vertices, we have to do something else.

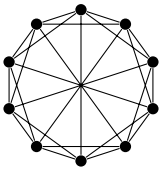
Fortunately, we do have a very flexible family of regular graphs at our fingertips: the circulant graphs introduced in Chapter 2. These are actually even more flexible than we need: for large n, we can pick from many examples of circulant graphs for each possible degree. To remind you of the definition, the circulant graph $\mathrm{Ci}_n(d_1, d_2, \ldots, d_k)$ has vertex set $\{0, 1, \ldots, n\}$, which we think of as being arranged around a circle, in that order. Two vertices x and y are adjacent if $x - y \equiv \pm d_i \pmod{n}$ for some i; if x and y are d_i steps apart around the circle.

Usually, each offset d_i gives each vertex x two neighbors: $x - d_i \bmod n$ and $x + d_i \bmod n$. As a result, if r is even, we can get an r-regular circulant graph by using $\frac{r}{2}$ offsets. To give a concrete example, $\mathrm{Ci}_n(1, 2, \ldots, \frac{r}{2})$ will be an r-regular graph for all even r between 2 and $n-1$, proving almost half of the theorem by itself. (For an example, see Figure 5.2a: this is the graph we use when $n = 9$ and $r = 6$.)

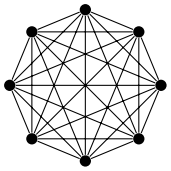
Question:	What if $r = 0$?
Answer:	I am reluctant to write $\mathrm{Ci}_n()$, because this looks weird, but it's true that if we don't give a circulant graph any offsets, then there will be no edges: the result is a 0-regular graph, for any positive integer n.



(a) $\mathrm{Ci}_9(1,2,3)$



(b) $\mathrm{Ci}_{10}(1,2,5)$



(c) $\mathrm{Ci}_8(1,2,3,4)$

Figure 5.2: Circulant graphs in the proof of Theorem 5.1

It is a bit trickier to get an r-regular circulant graph when r is odd. We know that it is impossible to find an r-regular graph on n vertices if both r and n are odd, so we may assume that n is even. But how does this help us?

Question:	In what exceptional case does an offset d_i in a circulant graph not contribute $+2$ to the degree of every vertex?
Answer:	When n is even and the offset is equal to $\frac{n}{2}$.

For any x, $x - \frac{n}{2} \bmod n$ and $x + \frac{n}{2} \bmod n$ are the same vertex: if we think of the vertices as being arranged around a circle, then this is the vertex opposite x. So including the offset $\frac{n}{2}$ lets us obtain an r-regular graph when r is odd. Again, we must give a concrete example to make the proof precise about it does, so let our example be $\mathrm{Ci}_n(1, 2, \ldots, \frac{r-1}{2}, \frac{n}{2})$. (For an example, see Figure 5.2b: this is the graph we use when $n = 10$ and $r = 5$.)

Question:	Why is this an r-regular graph?
Answer:	For each $i = 1, \ldots, \frac{r-1}{2}$, a vertex x is adjacent to $x + i \bmod n$ and $x - i \bmod n$, giving it $2 \cdot \frac{r-1}{2}$ or $r - 1$ neighbors. The last neighbor is $x + \frac{n}{2} \bmod n$.

Because $r \leq n - 1$, it is always true that $\frac{r-1}{2} < \frac{n}{2}$, so the initial progression $1, 2, \ldots, \frac{r-1}{2}$ never risks overlapping the exceptional case $r = n$. Similarly, in our previous case when r was even, the offsets $1, 2, \ldots, \frac{r}{2}$ never included the "exceptional" offset $\frac{n}{2}$, because $\frac{r}{2} < \frac{n}{2}$.

This completes the proof: for all r and n such that $0 \leq r \leq n-1$ and at least one of r or n is even, either the circulant graph $\mathrm{Ci}_n(1, 2, \ldots, \frac{r}{2})$ (for even r) or the circulant graph $\mathrm{Ci}_n(1, 2, \ldots, \frac{r-1}{2}, \frac{n}{2})$ (for odd r) is an r-regular graph on n vertices. □

Question:	What graph do we end up constructing when $r = n - 1$?
Answer:	For odd n, we get $\mathrm{Ci}_n(1, 2, \ldots, \frac{n-1}{2})$, and for even n, we get $\mathrm{Ci}_n(1, 2, \ldots, \frac{n}{2})$, but in both cases, the result is isomorphic to the complete graph K_n. This is because in both cases, we've included every possible offset. For an example of this, see Figure 5.2c.

I chose the circulant graphs I did in the proof of Theorem 5.1 not just because they're the simplest possible choice, but for another reason. The r-regular circulant graph on n vertices that we used has a special name: the *Harary graph*, denoted $H_{n,r}$. For example, the graphs in Figure 5.2 are $H_{9,6}$ (Figure 5.2a), $H_{10,5}$ (Figure 5.2b), and $H_{8,7}$ (Figure 5.2c).

We will see these graphs again in Chapter 26, where we will put them to the same use as their inventor Frank Harary did in 1962 [49]: as examples of graphs of a given connectivity with as few edges as possible.

5.3 How many regular graphs are there?

Though Theorem 5.1 proves the existence of r-regular, n-vertex graphs with all compatible values of n and r, we've only found one graph for each pair (n, r). How many possibilities exist in total? Let's start with some small values of r and see how the story unfolds.

When $r = 0$, every vertex is an isolated vertex: it has degree 0. There cannot be any edges. For any possible vertex set, there is exactly one graph, sometimes called the empty graph.

When $r = 1$, whether we believe in one possibility or multiple possibilities depends on how we count. You see, suppose x is a vertex in a 1-regular graph. Then x has only one neighbor, which we can call y. Vertex y also has only one neighbor, and we already know what it is: x. Thus, x and y form a 2-vertex connected component with each other and nothing else, and the entire graph is split up into $n/2$ such components. (A 1-regular graph, as we already know, is only possible when n is even, so $n/2$ is an integer.)

The counting difficulty comes in when we ask how many ways there are to choose these components. In one sense, there are many options: Figure 5.3 shows three different possibilities in just the 4-vertex case, and the total number of possibilities grows quickly. However, they are all isomorphic to each other. We say that there are 3 different 1-regular graphs with vertex set $\{1, 2, 3, 4\}$, but that up to isomorphism, there is only one 1-regular graph with 4 vertices. This is a term we'll use often, so let me give it a formal definition:

Definition 5.3. *A description of a graph G* ***up to isomorphism*** *is a description which is not necessarily true of G, but is true of some graph isomorphic to G.*

Here are some of the ways this phrase is used, in other circumstances.

- We say that the solution to a problem is unique up to isomorphism when all solutions to the problem are isomorphic graphs.
- We say that S is the set of all graphs of some type, up to isomorphism, when every graph of that type is isomorphic to some graph in S, and no two graphs in S are isomorphic. If $|S| = k$, we might also say that there are k graphs of that type, up to isomorphism.
- We say that the result of an operation is some particular graph G, up to isomorphism, if the operation results in a graph isomorphic to G, but possibly not with the same vertex set as G.

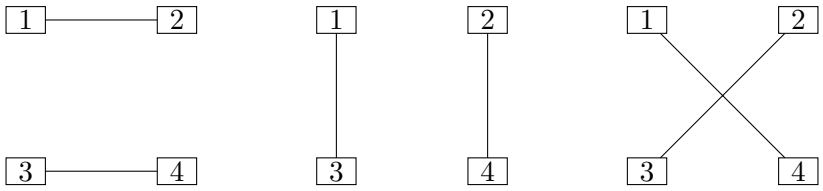
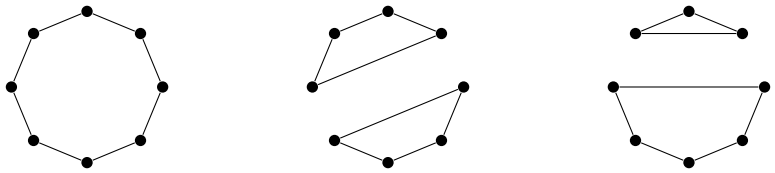


Figure 5.3: Three different(?) 1-regular graphs with vertex set $\{1, 2, 3, 4\}$



(a) C_8 (b) Two copies of C_4 (c) Copies of C_3 and C_5

Figure 5.4: Three non-isomorphic 2-regular graphs with 8 vertices

Statements up to isomorphism are also sometimes phrased in terms of *unlabeled graphs*, which is a perspective I'll say more about in Chapter 11, but I prefer to avoid it, because it causes more confusion. For any graph G, the vertex set $V(G)$ is a set of distinguishable objects, whether or not we distinguish them in a diagram of G.

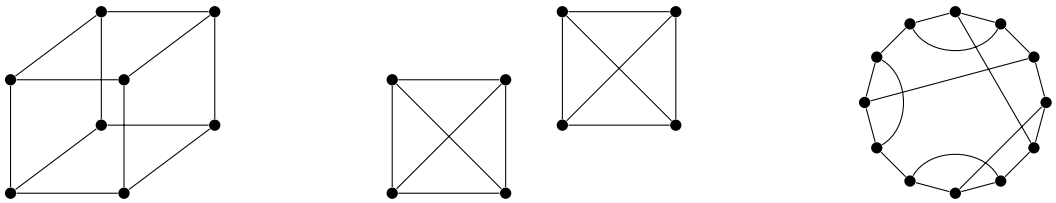
Moving on, let's consider what happens when $r = 2$. The example we constructed in Theorem 5.1 is $\mathrm{Ci}_n(1)$, but this is isomorphic to a more common graph: the cycle graph C_n. This is also unique in a sense, but a slightly different sense:

Proposition 5.2. *Up to isomorphism, C_n is the unique connected 2-regular graph with n vertices.*

Proof. First of all, we check that C_n a connected 2-regular graph. In C_n (with vertex set $\{1, 2, \ldots, n\}$) vertex x is adjacent to $x - 1$ and $x + 1$, treating $1 - 1$ as n and $n + 1$ as 1, so $\deg(x) = 2$ for all $x \in V(C_n)$. For all $x, y \in V(C_n)$, the sequence $(x, x+1, \ldots, y)$ is an $x-y$ walk if $x < y$, and the sequence $(x, x - 1, \ldots, y)$ is an $x - y$ walk if $x > y$; this makes C_n connected.

Next, let G be an arbitrary connected 2-regular graph. Then in particular G has minimum degree 2, so Theorem 4.4 applies and tells us that G contains a cycle C. For all $x \in V(C)$, we know $\deg_C(x) = 2$ because C is a cycle, and $\deg_G(x) = 2$ because G is 2-regular, so x has no neighbors in G other than the two it has in C. There are no edges with exactly one endpoint in $V(C)$, so by Lemma 3.2, either $G = C$ or else G wouldn't be the connected graph we took it to be. But $G = C$ makes G isomorphic to a cycle graph, completing our proof. □

The difference between the 1-regular and 2-regular case is that we can obtain more examples by taking graphs with multiple connected components. Each component, by Proposition 5.2, is a cycle; we can mix and match cycles however we like to reach a total of n vertices. Figure 5.4 shows three non-isomorphic possibilities for an 8-vertex 2-regular graph: it could be isomorphic to C_8 (Figure 5.4a), or to the union of two copies of C_4 (Figure 5.4b), or to the union of copies of C_3 and C_5 (Figure 5.4c).



(a) Q_3 (b) Two copies of K_4 (c) The Frucht graph

Figure 5.5: Three examples of 3-regular graphs

Once $r \geq 3$, any hope of classifying the r-regular graphs on n vertices goes out the window. Even when $r = 3$, there are many examples.

Figure 5.5 shows several 3-regular graphs. You will recognize the graph in Figure 5.5a: it is the cube graph Q_3. The graph in Figure 5.5b should not be too much of a surprise either: it is the disjoint union of two copies of K_4, and is the smallest 3-regular graph that is not connected.

You might be forgiven for thinking, based on the examples you've seen, that regular graphs are highly symmetric—after all, every vertex has the same degree, which makes every vertex like every other, at least superficially. To dissuade you from that impression, I've included a final example in Figure 5.5c: the Frucht graph. It is named after Robert Frucht, who discovered it in 1949 [35]. The Frucht graph is notable for being completely asymmetric; in other words, it has no automorphisms, other than the identity automorphism!

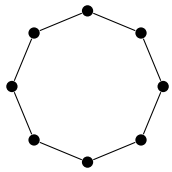
The Frucht graph is just the smallest graph with this property. When n is large, it is very rare for a 3-regular graph with n vertices to have a nontrivial automorphism. Don't be misled by graphs like circulant graphs, which have a lot of symmetry—they are just the graphs it is easiest to point to, precisely for that reason. (Think about how much effort it would be to describe the Frucht graph exactly, compared to how easy it is to define the circulant graph $\mathrm{Ci}_{12}(1, 6)$, which has the same number of vertices and edges.)

The Online Encyclopedia of Integer Sequences (OEIS) lists integer sequences with notable mathematical properties. Entry A002851 of the OEIS [80] gives the number of connected 3-regular graphs on $2n$ vertices, up to isomorphism. I cite an online encyclopedia rather than give a formula because there is no clean formula for the entries of this sequence. However, you can observe that they grow rather quickly from the first few entries. Starting from $2n = 4$, the sequence is

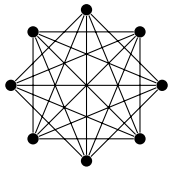
$$1, 2, 5, 19, 85, 509, 4060, 41301, 510489, 7319447, \ldots$$

You can probably guess that the story for 4-regular graphs, 5-regular graphs, and so on is very similar. We only regain any semblance of order once r gets very close to n. We discover that semblance of order by taking the complement graph: toggling all the edges from present to absent and vice versa.

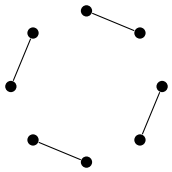
Question:	If G is an r-regular graph on n vertices, what can you say about the degrees in its complement $\overline{G}$?
Answer:	Every vertex $x \in V(G)$ has r neighbors in G, out of $n - 1$ other vertices, so its neighbors in $\overline{G}$ are the $(n - 1) - r$ vertices it was not previously adjacent to. Therefore $\overline{G}$ is $(n - r - 1)$-regular.



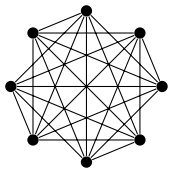
(a) C_8



(b) $\overline{C_8}$



(c) A 1-regular graph



(d) A 6-regular graph

Figure 5.6: Regular graphs and their complements

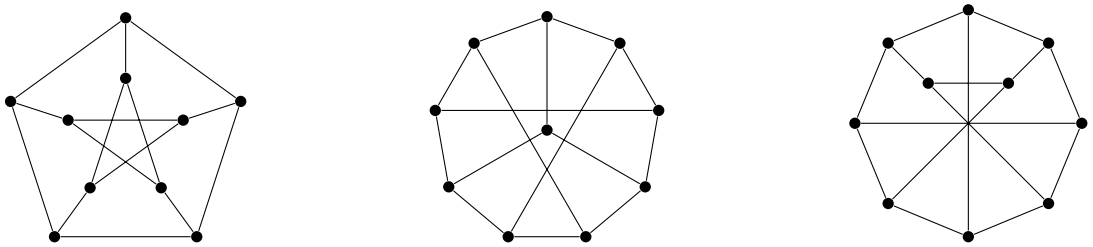


Figure 5.7: Three diagrams of the Petersen graph

Figure 5.6 shows some examples of regular graphs and their complements. For example, the cycle graph C_8 is an 8-vertex 2-regular graph we understand, so we must also be able to understand its complement, an 8-vertex 5-regular graph. We already understand the complete graph K_n quite well; it is the only n-vertex $(n-1)$-regular graph, up to isomorphism, becuase if every vertex has degree $n-1$, then it is adjacent to all $n-1$ other vertices. In fact, the empty graph we previously took as our 0-regular example is commonly written $\overline{K_n}$, taking the complete graph as a starting point.

Up to isomorphism, there is only one 1-regular graph on n vertices, and only when n is even. Taking its complement, we learn that up to isomorphism, there is only one $(n-2)$-regular graph on n vertices, and only when n is even. Figures 5.6c and 5.6d show an example of this.

Among 2-regular graphs, cycles are special; they are the only connected graphs. Among $(n-3)$-regular graphs, the complements of cycles are not as special, because (for $n \geq 5$) all $(n-3)$-regular graphs are connected. Still, by taking the complement of every graph in Figure 5.4, we could find all the 5-regular 8-vertex graphs, up to isomorphism; the same strategy works for all n.

5.4 The Petersen graph

Having seen a bit of the variety of regular graphs out there, we'll now look at another useful family of regular graphs. We'll begin with a specific example, known as the Petersen graph after its discoverer, Julius Petersen.

The Petersen graph is notable in graph theory for many reasons. In this textbook, we will see it several times in its role of providing a small counterexample to several plausible-sounding conjectures in graph theory. There are also several classification theorems (which we will not see) in which it plays a central role. Entire books [57] have been written about this graph.

Three diagrams of the Petersen graph are shown in Figure 5.7, showing some of the symmetry it has: one with 5-fold rotational symmetry, one with 3-fold symmetry, and one which nearly has 4-fold symmetry (but not quite). In fact, the Petersen graph is much more symmetric than any of these diagrams show, which can be seen from its combinatorial definition:

Definition 5.4. *The **Petersen graph** is the graph with vertex set*

$$\{12, 13, 14, 15, 23, 24, 25, 34, 35, 45\}$$

consisting of all unordered pairs of elements of $\{1, 2, 3, 4, 5\}$*, and an edge between two vertices whenever they have no elements of* $\{1, 2, 3, 4, 5\}$ *in common. (For example, vertex* 12 *is adjacent to vertices* 34*,* 35*, and* 45*.)*

This definition has a much greater degree of symmetry than either of the diagrams! For any permutation of the set $\{1, 2, 3, 4, 5\}$, we can relabel the vertices of the Petersen graph according to that permutation; this will not change which vertices have no elements of $\{1, 2, 3, 4, 5\}$ in common, so it is an automorphism of the Petersen graph. All $5! = 120$ automorphisms of the Petersen graph can be described in this way.

The Petersen graph is part of an infinite family of regular, highly symmetric graphs known as the Kneser graphs. For all integers n and k with $0 \leq k \leq n$, the vertices of the *Kneser graph* $K(n, k)$ are the k-element subsets of the set $\{1, 2, \ldots, n\}$. As in the definition of the Petersen graph, two vertices of $K(n, k)$ are adjacent if they have no elements in common. (The definition is only interesting when $n \geq 2k$; otherwise, any two k-element subsets of $\{1, 2, \ldots, n\}$ are guaranteed to overlap.)

Question:	How many vertices does $K(n, k)$ have, as a function of n and k?
Answer:	There are $\binom{n}{k}$ vertices: the number of ways to choose k unordered elements of $\{1, 2, \ldots, n\}$ without repetition.

Question:	What is the degree of an arbitrary vertex of $K(n, k)$?
Answer:	A vertex X has $\binom{n-k}{k}$ neighbors: the k-element subsets of the set $\{1, 2, \ldots, n\} - X$.

This explains why the Kneser graphs are always regular. For the same reason as the Petersen graph, they have many automorphisms. Though the Petersen graph is by far the most widely encountered of the Kneser graphs, they all have applications to the combinatorics of set families, due to their definition via subsets of $\{1, 2, \ldots, n\}$.

For now (in this section), I will prove only one property of the Petersen graph, which might look silly on its own. However, it will be a starting point for other proofs, and in the next section, we will see one example of how it can be useful. This lemma is also a good example of how to use symmetry in a proof.

Lemma 5.3. *The Petersen graph has no cycles of length* 3 *or* 4*.*

Proof. To check such a claim with as little effort as possible, it helps to make use of the many automorphisms of the Petersen graph. Take any two adjacent vertices of the Petersen graph: these are unordered pairs ab and cd, where a, b, c, d are four distinct elements of $\{1, 2, 3, 4, 5\}$. Then there is an automorphism of the Petersen graph that relabels a to 1, b to 2, c to 3, and d to 4 in every vertex, taking our two adjacent vertices to 12 and 34.

How do we use this? Well, suppose C is an arbitrary cycle in the Petersen graph, and ab and cd are two vertices adjacent in C. By applying the automorphism above to C, we get a new cycle C' in the Petersen graph on which 12 and 34 are two consecutive vertices. The automorphism is a bijection, so C and C' have the same number of vertices: if we prove C' has length at least 5, we conclude that C has length at least 5.

(When writing for an audience familiar with such tricks, it is common to simply say: "Let C be a cycle in the Petersen graph. By symmetry, we may assume that 12 and 34 are two consecutive vertices along C." The proof becomes much shorter.)

On a cycle of length 3, two consecutive vertices have a common neighbor: the third vertex. On a cycle of length 4, two consecutive vertices each have another neighbor, and those two neighbors must themselves be adjacent. To complete the proof, we show that neither of these scenarios is possible in the Petersen graph, when 12 and 34 are the two consecutive vertices.

The other neighbors of 12 are 35 and 45; the other neighbors of 34 are 15 and 25. From this we can see that 12 and 34 have no common neighbors, so our cycle cannot have length 3; also, none of the neighbors of 12 and 34 are adjacent (all of them include the number 5), so our cycle cannot have length 4. □

5.5 The degree/diameter problem

In Chapter 3, we defined the distance $d_G(x, y)$ between two vertices in a graph G to be the minimum length of an $x - y$ walk in G. A related concept is the *diameter* of G: the largest distance between any pair of vertices.[9] That is, a graph has diameter D if we can reach every vertex from every other vertex in at most D steps, but there are some cases in which we cannot do it in fewer than D steps.

It's worth pointing out that the diameter is defined by a maximization problem (it is the largest distance) with a minimization problem inside it (the distance is the smallest possible length of a walk). This means that proving that the diameter of a graph has a certain value takes some getting used to: it is a mix of specific examples and universal bounds, in both directions.

Question:	How might you prove that a graph G has diameter at least D?
Answer:	By proving that $d_G(x, y) \geq D$ for two particular vertices x and y: showing that there are no $x - y$ walks of any length less than D.

[9] This term in graph theory is inspired by the diameter of a circle, since that is the largest distance between any two points in or on the circle.

Question:	How might you prove that a graph G has diameter at most D?
Answer:	By proving that $d_G(x, y) \leq D$ for all pairs of vertices x, y, showing that we can get from every vertex to every other in at most D steps.

We can see both kinds of proof in action when we prove the following result, which is also the first noteworthy use of the Petersen graph in this textbook.

Proposition 5.4. *Among all graphs with maximum degree* 3 *and diameter* 2*, the Petersen graph has the most vertices.*

Proof. First, we should verify that the Petersen graph actually does have diameter 2. The lower bound is quick: a diameter of 1 would mean that all vertices are adjacent, and the Petersen graph certainly has pairs of vertices that are not adjacent (such as 12 and 13). Therefore the Petersen graph has diameter at least 2.

For the upper bound, we must show that no matter which vertex x we choose in the Petersen graph, all vertices are within distance 2 of x. Since the Petersen graph is 3-regular, x has three neighbors; call them y_1, y_2, y_3. Each y_i has two neighbors other than x. By Lemma 5.3, y_i cannot be adjacent to another y_j (or else (x, y_i, y_j, x) would represent a cycle of length 3), and two neighbors y_i and y_j of x cannot have a common neighbor z (or else (x, y_i, z, y_j, x) would represent a cycle of length 4.) Therefore there are six different vertices $z_1, \ldots, z_6$ all adjacent to a neighbor of x.

Altogether, we've named 10 vertices: x, $y_1, \ldots, y_3$, and $z_1, \ldots, z_6$. All are different, and all are within distance 2 of x. But the Petersen graph only has 10 vertices, so we conclude that all vertices are within distance 2 of x. Since x was arbitrary, the diameter of the Petersen graph is at most 2.

The last part of the proof is the interesting part, in my opinion. We must prove that a graph G with maximum degree 3 and diameter 2 can have at most 10 vertices: the number of vertices in the Petersen graph. In such a graph, if we pick an arbitrary vertex x, we can write $V(G)$ as $\{x\} \cup Y \cup Z$, where Y is the set of vertices at distance 1 from x, and Z is the set of vertices at distance 2 from x.

The graph G has maximum degree 3, so in particular, x has at most 3 neighbors; every vertex in Y must be a neighbor of x, so $|Y| \leq 3$. Meanwhile, for every vertex $z \in Z$, there is a path (x, y, z) of length 2; here, $y \in Y$, because it is adjacent to x. Each vertex has at most 3 neighbors, but one of them is x: it has at most 2 neighbors in Z. Together, the vertices in Y have at most $3 \cdot 2 = 6$ neighbors in Z, but each vertex in Z must have a neighbor in Y: so $|Z| = 6$.

Putting this together, $|V(G)| = |\{x\}| + |Y| + |Z| \leq 1 + 3 + 6 = 10$, so G can have at most 10 vertices. The Petersen graph, with 10 vertices, is optimal. □

The degree/diameter problem studies the generalization of Proposition 5.4: for a pair of positive integers (Δ, D), what is the largest number of vertices in a graph with maximum degree Δ and diameter D?

Theorem 5.5. *A graph G with maximum degree $\Delta \geq 2$ and diameter D can have at most*

$$1 + \Delta + \Delta(\Delta-1) + \Delta(\Delta-1)^2 + \cdots + \Delta(\Delta-1)^{D-1}$$

vertices, which is $2D+1$ if $\Delta = 2$ and $1 + \Delta\frac{(\Delta-1)^D-1}{\Delta-2}$ if $\Delta > 2$.

Proof. As in the proof of Proposition 5.4, we pick an arbitrary vertex x and write the vertex set of G as

$$V(G) = Y_0 \cup Y_1 \cup Y_2 \cup \cdots \cup Y_D$$

where $Y_0 = \{x\}$ and Y_i is the set of all vertices at distance i from x.

Next, by induction on i, we show that $|Y_i| \leq \Delta(\Delta-1)^{i-1}$ for $1 \leq i \leq D$. When $i = 1$, this inequality says that $|Y_1| \leq \Delta$, which is true because x has at most Δ neighbors, and every vertex in Y_1 is a neighbor of x. This proves the base case.

An important property we'll need before we continue is that for $1 \leq i \leq D$, a vertex $y \in Y_i$ must have a neighbor in Y_{i-1}. That's because there is an $x - y$ walk of length i in G. Let z be the next-to-last vertex on that walk: the walk is $(x, \ldots, z, y)$. This z will be the neighbor of y in Y_{i-1}.

Question:	Why is z a neighbor of y?
Answer:	By the definition of a walk, yz must be an edge.

Question:	Why is there an $x - z$ walk of length $i-1$?
Answer:	Remove the last vertex y from the walk $(x, \ldots, z, y)$ to get such an $x - z$ walk.

Question:	Why is there no $x - z$ walk of length $i-2$ or less?
Answer:	We could take such a walk and add y to the end, getting an $x - y$ walk of length $i-1$ or less; this contradicts $y \in Y_i$.

Now we're ready for the induction step. Suppose that for $1 \leq i \leq D-1$, we have already shown that $|Y_i| \leq \Delta(\Delta-1)^{i-1}$, and want to move on to $|Y_{i+1}|$. Each vertex in Y_i has at least one neighbor in Y_{i-1}, so it has at most $\Delta - 1$ neighbors in Y_{i+1}. Together, there are at most $\Delta(\Delta-1)^{i-1}$ vertices in Y_i, so they have at most $\Delta(\Delta-1)^i$ neighbors in Y_{i+1}. But every vertex in Y_{i+1} must be such a neighbor, so $|Y_{i+1}| \leq \Delta(\Delta-1)^i$, completing the induction step.

Once the induction is concluded, we have

$$\begin{aligned} |V(G)| &= |Y_0| + |Y_1| + |Y_2| + \cdots + |Y_D| \\ &= 1 + \underbrace{\Delta}_{|Y_1|} + \underbrace{\Delta(\Delta-1)}_{|Y_2|} + \underbrace{\Delta(\Delta-1)^2}_{|Y_3|} + \cdots + \underbrace{\Delta(\Delta-1)^{D-1}}_{|Y_D|}, \end{aligned}$$

which is exactly the bound we wanted. When $\Delta = 2$, the bound on $|Y_i|$ simplifies to 2 for every i, giving us $2D+1$ for the sum. When $\Delta > 2$, the formula $1 + \Delta\frac{(\Delta-1)^D-1}{\Delta-2}$ comes from the formula for the sum of a finite geometric series: $a + ar + ar^2 + \cdots + ar^n = a\frac{r^{n+1}-1}{r-1}$. □

The upper bound in Theorem 5.5 is called the Moore bound after Edward Moore (the same Moore that discovered the distance-finding algorithm discussed in Chapter 3). Moore also posed the problem of finding the graphs which achieve the bound exactly, and such graphs are known as Moore graphs. For example, Proposition 5.4 tells us that the Petersen graph is a Moore graph. In order for the inequalities of Theorem 5.5 to be equations, Moore graphs must all be regular graphs: the maximum degree Δ must actually be the degree of every vertex.

Moore graphs are very rare, apart from a few initial cases: they exist for all $\Delta \geq 2$ when $D = 1$, and for all $D \geq 1$ when $\Delta = 2$. I will leave it to you, in the practice problems, to understand these two constructions. Moore graphs were first studied by Alan Hoffman and Robert Singleton in 1960 [56], who found the Petersen graph for $(\Delta, D) = (3, 2)$ and a graph called the Hoffman–Singleton graph for $(\Delta, D) = (7, 2)$. What's more, they proved that when $D = 2$ or $D = 3$, and $\Delta \geq 3$, there are no more Moore graphs—with one possible exception. Hoffman and Singleton were unable to determine if there is a Moore graph with $\Delta = 57$ and $D = 2$; to this day, we do not know the answer!

Since 1960, the degree/diameter problem has advanced considerably; a 2013 survey by Mirka Miller and Jozef Širáň [73] summarizes much of the recent progress. We now know that, aside from the two graphs found by Hoffman and Singleton, and the possible $(\Delta, D) = (57, 2)$ case, there are no Moore graphs with $\Delta \geq 3$ and $D \geq 2$. In most cases, the best constructions we know are very far from the upper bound of Theorem 5.5. Many, but not all of them are regular graphs.

5.6 Practice problems

1. What are the possible values of the diameter of an 8-vertex graph? Give an example for each possible value.

2. If n and r are both odd, then an r-regular graph on n vertices does not exist, but a Harary graph $H_{n,r}$ still does. In this case, $H_{n,r}$ is a nearly-regular graph: it is a graph on n vertices where $n - 1$ of them have degree r, and one has degree $r + 1$.

   It is defined starting from the Harary graph $H_{n,r-1}$, or $\mathrm{Ci}_n(1, 2, \ldots, \frac{r-1}{2})$. To this graph, we add $r + 1$ edges that increase the degree of vertex 0 by 2, and the degree of vertices $1, 2, \ldots, n - 1$ by 1 each.

   How can we do this? Prove that your method works in general.

3. Here is a fragment of an alternate proof of the $r = 3$ case of Theorem 5.1.

   ... assume that a 3-regular graph H on $n - 4$ vertices exists. Then, we can create a 3-regular graph G on n vertices, just by adding a new connected component to H: four new vertices adjacent to each other and to no other vertices ...

   What kind of a proof is this? What else do we need to do to finish the proof of Theorem 5.1 using this idea?

4. For each diagram in Figure 5.7, show how to label the vertices with elements of the set $\{12, 13, \ldots, 45\}$ (the vertex set of the Petersen graph) so that two vertices are adjacent in the diagram if and only if their labels have no digit in common.

5. Prove that for all $n \geq 5$, the complement of the path graph P_n has diameter 2.

6. a) Prove that the Kneser graph $K(n, k)$ is connected when $n > 2k$.

   b) Find and prove a similar condition that determines when $K(n, k)$ has diameter 2.

   c) Find and prove a similar condition that determines when $K(n, k)$ has no cycles of length 3 or 4.

7. a) For all $\Delta \geq 2$, there is a Moore graph with maximum degree Δ and diameter 1 (and $\Delta + 1$ vertices). What is it?

   b) For all $D \geq 1$, there is a Moore graph with maximum degree 2 and diameter D (and $2D + 1$ vertices). What is it?

8. Prove that, just like the Petersen graph, the Hoffman–Singleton graph has no cycles of length 3 or 4. (You do not know very much about the Hoffman–Singleton graph, but all you need to know for this problem is there in this chapter.)

9. Let $\Delta \geq 3$. Prove that if a graph G with maximum degree $\Delta \geq 2$ and diameter D is not regular, then it has at most

$$1 + (\Delta - 1) + (\Delta - 1)^2 + \cdots + (\Delta - 1)^{D-1} = \frac{(\Delta - 1)^D + 1}{\Delta - 2}$$

   vertices: approximately $\frac{\Delta - 1}{\Delta}$ of the Moore bound.

10. (BMO 1972) There are n persons present at a meeting. Every two persons are either friends of each other or strangers to each other. No two friends have a friend in common. Every two strangers have two and only two friends in common.

    a) Prove that each person has the same number of friends at the meeting.

    b) If that number is 5, find n.

6 Graphic sequences

The purpose of this chapter

Most graph theory textbooks present the graphic sequence algorithm differently, by working with numerical sequences rather than directly with graphs. In my opinion, that makes the problem harder as well as more boring. This leaves much less class time for discussing edge swaps and the ideas in the proof of the Havel–Hakimi theorem (Theorem 6.2).

I think that Problem 6.1 and its solution for 8 people is a good way to motivate the graphic sequence algorithm. The proof of Proposition 6.1 could be skipped in a graph theory course covering this material, but I felt that I had to include it to avoid leaving the problem without a solution.

6.1 An unusual party

Let me tell you a story.[10] One time, I was at a very large party—there were 100 people there! Not everybody knew each other, of course. In fact, each person at the party knew a different number of people. In addition, I—

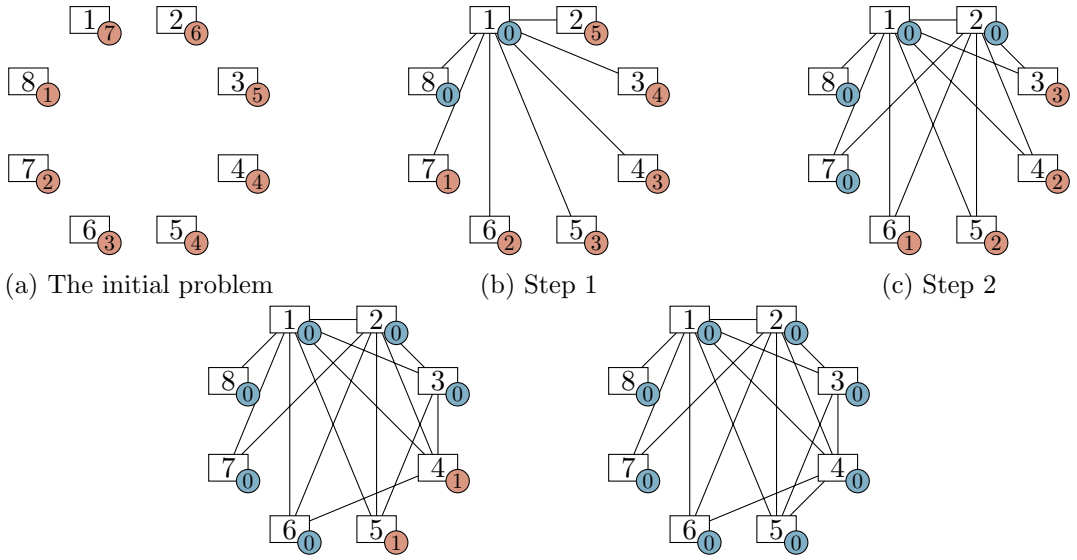
"—That can't be right!" you interrupt. "You're saying that the graph which represents who knew each other at the party had the degree sequence $99, 98, 97, \ldots, 2, 1, 0$? But that's not a graphic sequence!"

Question:	Why isn't this sequence graphic?
Answer:	The vertex of degree 99 is adjacent to every other vertex, but the vertex of degree 0 is adjacent to none of the other vertices, and these facts contradict each other.

Oh, sorry. My mistake. Let me try again.

Problem 6.1. *One time, I was at a very large party—there were* 100 *people there! Not everybody knew each other, of course. In fact, each person at the party knew a different number of people, with one exception: I personally knew the same number of people as another guest, named Masha. In addition, I can promise you that everyone at this party knew at least one other person.*

[10] This appears to be a very common story to tell. In West's *Introduction to Graph Theory* [104], a version of it appears as "The Handshake Problem", and another version was a problem in the 1985 British Mathematical Olympiad.



(a) The initial problem (b) Step 1 (c) Step 2

(d) Step 3 (e) Step 4

Figure 6.1: Reconstructing the 8-person party

Now, given this information, can you tell me how many people I knew at the party? Also, can you tell me whether Masha and I knew each other?

Before we solve this problem, let's consider a smaller case: an 8-person party and its graph of who knows whom. In this graph, the vertex degrees are $7, 6, 5, 4, 3, 2, 1$, but with one of these duplicated. We can't duplicate an odd number, because then the sum of degrees would be odd, in violation of Corollary 4.2 to the handshake lemma. So we must duplicate the 6, the 4, or the 2. I can tell you the correct option: duplicating the 4 is the only choice that will work.

If the degree sequence is $7, 6, 5, 4, 4, 3, 2, 1$, can we reconstruct the graph? Let me show you a way to think about it with the aid of a diagram. In Figure 6.1a, I have drawn vertices 1 through 8, each with an extra label: the degree that the vertex wants to have. As we work on filling in the graph, we will update these labels to show the remaining number of incident edges that the vertex wants, but doesn't have yet.

For one of these vertices, we can immediately assign the neighbors. Vertex 1 needs to be adjacent to every other vertex, so we can draw all of those edges, as shown in Figure 6.1b. This reduces the label on vertex 1 by 7 (we've given it 7 more incident edges), and the label on each other vertex by 1 (we've given each of them 1 incident edge). It's important to point out that vertex 1 and 8 now want 0 more edges, so they will not be used again.

We can continue. Vertex 2 needs 5 more edges, and there are only 5 other vertices that can accept edges, so it must be adjacent to all of them! We add all of those edges in Figure 6.1c. Updating the label on each vertex, we see that vertices 2 and 7 are also satisfied.

Two more steps solve the problem entirely. When we process vertex 3 in Figure 6.1c, we arrive at the diagram in Figure 6.1d. At that point, we only have two vertices left to deal with: vertices 4 and 5. Each one wants 1 more neighbor, so we make them adjacent, arriving at the diagram in Figure 6.1e. Now we know all the edges of the graph.

We will return to the ideas in this solution very soon. For now, let me make one more observation. In Figure 6.1b, there are 6 vertices left to deal with (vertices 2 through 7), and they demand $5, 4, 3, 3, 2, 1$ additional edges respectively. This is a 6-vertex version of the same problem, which suggests a solution to Problem 6.1 by induction.

Proposition 6.1. *For all $n \geq 1$, up to isomorphism, there is a unique $2n$-vertex graph which contains a vertex of every degree $1, 2, \ldots, 2n-1$. In that graph, there are two vertices of degree n, and they are adjacent.*

Proof. We induct on n. When $n = 1$, the graph we are looking for is a graph with 2 vertices, at least one of which has degree 1. There must be an edge between the 2 vertices, and this already describes the graph uniquely. In this graph, there are two vertices of degree 1, and they are adjacent, completing the base case.

Next, for some $n > 1$, assume that the previous case of the proposition holds. That is, there is a unique $2(n-1)$-vertex graph G which contains a vertex of degree $1, 2, \ldots, 2n-3$. In G, there are two vertices of degree $n-1$, and they are adjacent.

Mimicking the way in which the 6-person party sits inside the 8-person party in Figure 6.1b, we extend G to a $2n$-vertex graph H by adding two vertices x and y; we make x adjacent to y and to all vertices of G, and we make y adjacent only x. Here, $\deg_H(x) = 2n-1$ and $\deg_H(y) = 1$; for all vertices $z \in V(G)$, we have $\deg_H(z) = \deg_G(z) + 1$ (because we added the edge xz) and so the vertices of G fill in the degrees $2, 3, \ldots, 2n-2$. The two vertices of degree $n-1$ in G become vertices of degree n in H, and they are adjacent.

This proves that the situation in Proposition 6.1 is possible, but not that it is unique. To prove it is unique, let H' be any $2n$-vertex graph that satisfies the conditions. Let x be a vertex of degree $2n-1$ in H', and let y be a vertex of degree 1 in H'. Then x must be adjacent to all other vertices (including y), which means y is only adjacent to x. Let $G' = H' - x - y$; then in G', every degree k between 1 and $2n-3$ is present, because degree $k+1$ is present in H on a vertex that's neither x nor y. Since G' satisfies the conditions in the proposition, it must be isomorphic to the graph G in the previous paragraph, which forces H' to be isomorphic to the graph H in the previous paragraph.

Question:	How do we know every isomorphism between G and G' extends to an isomorphism between H and H'?
Answer:	To extend an isomorphism $\varphi\colon V(G) \to V(G')$, just define $\varphi(x) = x$ and $\varphi(y) = y$. Since in both H and H', x is adjacent to every vertex and y is adjacent only to x, the extended φ is guaranteed to preserve the edges out of x and y.

This proves uniqueness, which completes the induction step; by induction, the graph exists and is unique up to isomorphism for all n. □

In particular, the answer to Problem 6.1 (the $n = 50$ case of Proposition 6.1) is that Masha and I both knew 50 people at the party, and we did know each other.

6.2 A graphic sequence algorithm

The reconstruction in Figure 6.1 is only possible for very special degree sequences. Our decision at each step was forced, and the reason it was forced is the uniqueness result in Proposition 6.1. Most degree sequences do not have a corresponding uniqueness result, in which case, life is more complicated.

What do we do if we're faced with a diagram like one of the ones in Figure 6.1, but there are no forced deductions to make? It turns out that there's still something we can do. It will feel like making a lucky guess. Later in this chapter, in Theorem 6.2, we will prove that the guess is always justified, at least in one sense. Before we do that, though, let me explain the guess and why it is at least reasonable.

My reasoning is this: in Figure 6.1, we were always living life on the edge. The highest-degree vertex always just barely had enough edges we could place, and that's why the result was unique. We would like our graphs to be as little like that, actually: we want to avoid having vertices which need many edges. We also want to avoid creating too many inactive vertices (which don't need any more edges) too quickly.

Putting these together, which vertex should we deal with next, at each step? It should be the vertex which is most in danger: the vertex with the highest remaining degree. Which neighbors should we give it, if we have the choice? It should be the other vertices with the highest remaining degree—this ensures those vertices need fewer edges later, and also avoids creating inactive vertices too early.

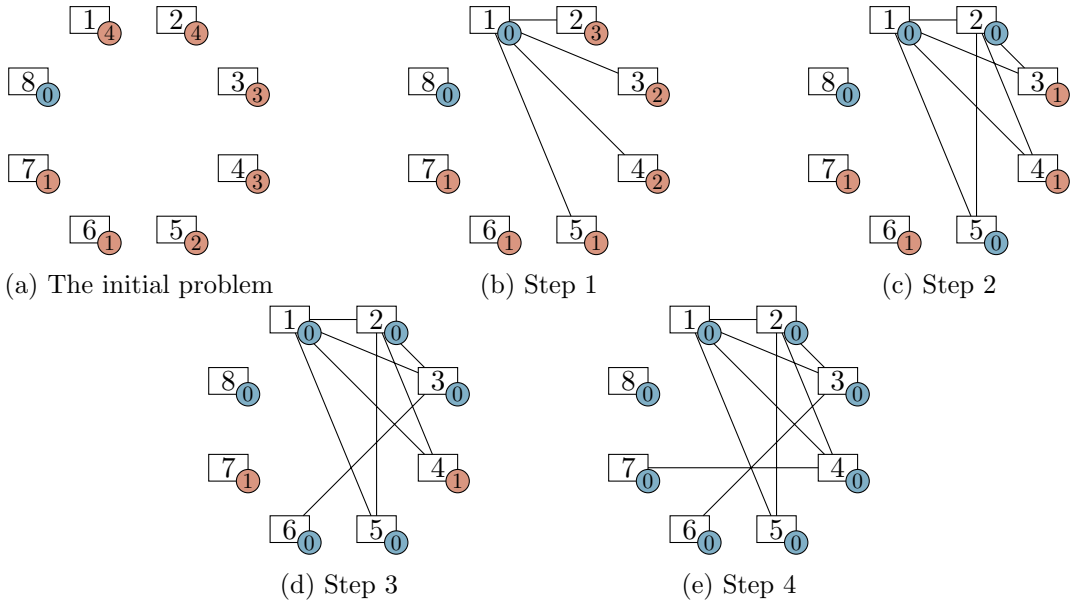
Now, let me present an algorithm which makes all these ideas formal and precise.

Given a sequence of nonnegative integers $d_1, d_2, \ldots, d_n$, our goal is to test if the sequence is graphic, and if it is, to construct a graph whose degree sequence is $d_1, d_2, \ldots, d_n$. We begin our construction with a graph G that has vertex set $V(G) = \{1, 2, \ldots, n\}$ and edge set $E(G) = \varnothing$: no edges to start with. We will add edges to G as we go.

For all $i = 1, \ldots, n$, we intend for vertex i to have degree d_i at the end. We give vertex i a *demand* representing how many more edges it needs to meet this goal. Initially, we set its demand equal to d_i, but we will update the demand of i over the course of the algorithm. We will call a vertex *active* if its demand is positive, and *inactive* if its demand is 0. In diagrams, I will give each vertex an extra label showing its demand.

We repeat the algorithm below for as long as any active vertices remain. In each iteration, we perform the following steps:

1. Pick a vertex x with the highest possible demand, breaking ties arbitrarily; let k be the demand of x.
2. If there are fewer than k active vertices other than x, stop and declare that the sequence is not graphic: we have failed to construct a graph with this sequence.
3. Otherwise, let S be a set of k active vertices, other than x, whose demand is as high as possible (again, breaking ties arbitrarily).
4. Add the k edges xy for all $y \in S$ to the graph G. Set the demand of x to 0 (making it inactive) and decrease the demand of each vertex $y \in S$ by 1.



(a) The initial problem (b) Step 1 (c) Step 2

(d) Step 3 (e) Step 4

Figure 6.2: Finding a graph with degree sequence $4, 4, 3, 3, 2, 1, 1, 0$

Once no more active vertices remain, we stop.

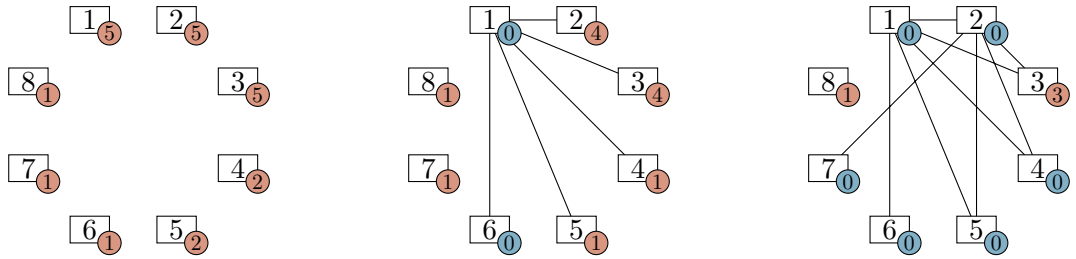
Question:	What is the maximum number of iterations we will need?
Answer:	Since at least one vertex becomes inactive at each step, we can be be certain to stop after n iterations (in an n-vertex graph), though we may stop sooner.

Question:	How do we know that when we construct a graph at the end, it has the right degree sequence?
Answer:	At each step, the degree of each vertex increases by the same amount that its demand decreases. Since the demand of vertex i decreases from d_i to 0 by the end, its degree must have increased from 0 to d_i: the value we wanted.

Let me show you two examples of this algorithm in action. After those examples, it will be more clear what we do and do not need to prove in order to be certain that the algorithm works.

6.3 Two examples

First, let's use the algorithm in the previous section to find a graph with the degree sequence $4, 4, 3, 3, 2, 1, 1, 0$. (If you have a good memory, you can be certain that at least one such graph exists, because we saw such a graph in Chapter 4.)



(a) The initial problem (b) Step 1 (c) Step 2

Figure 6.3: Trying to find a graph with degree sequence $5, 5, 5, 2, 2, 1, 1, 1$

We begin with a graph with no edges in which the demand of each vertex is equal to the degree we want it to have, as shown in Figure 6.2a. (Vertex 8 starts out inactive; it does not affect the problem in any way.)

We begin one of the vertices with demand 4 (I choose 1, though it is also possible to choose 2) and add edges to the 4 other vertices with the highest demands (vertices 2, 3, 4, and 5). The result is Figure 6.2b, with one fewer active vertex.

Our next choice is unique: vertex 2, with demand 3. The vertices that will become its remaining neighbors must include vertices 3 and 4 (with demand 2), as well as one of the vertices with demand 1; we arbitrarily choose it to be vertex 5. Adding the edges between these vertices and decreasing demand accordingly, we obtain Figure 6.2c.

Here, all four remaining vertices are tied for highest demand: their demands are all 1. Our next step will be to draw an edge between two of its vertices, and any of the ways to do that are fine. In Figure 6.2d, we add edge 36, leaving only two active vertices, and in Figure 6.2e, we add edge 47 and finish the problem. We can check our answer by checking that the label on each vertex in Figure 6.2a is equal to the degree it has in Figure 6.2e.

In this example, the sequence we started with was graphic; let's consider an example where the sequence we start with is not graphic. I will use the sequence $5, 5, 5, 2, 2, 1, 1, 1$.

The first step, going from Figure 6.3a to Figure 6.3b, is completely normal. We pick vertex 1 (we could also have picked vertex 2 or 3, since all of them have demand 5) and join it to vertices 2 through 6 (the 5 vertices with the highest demands). Vertices 1 and 6 become inactive at this step.

Next, we pick one of the vertices with demand 4: say, vertex 2. We draw 4 edges, one of which must go to 3, and the rest must go to vertices with demand 1. There are several ways to draw the edges, but all of them have a similar result to what is shown in Figure 6.3c. Almost all of the vertices have become inactive.

We can no longer proceed! In Figure 6.3c, vertex 3 has demand 3, but we do not have 3 other active vertices to connect to. So we declare that the sequence $5, 5, 5, 2, 2, 1, 1, 1$ is not graphic, and give up.

Now, where in this process do we need additional proof?

Nothing more is needed with the first example. In principle, once we have a graph with the right degree sequence, we don't care if we got it by a valid algorithm, or an invalid algorithm, or if it came to us in a dream: we have the answer, and we can check it! But we are in even

better shape than that: we've already shown that if the graphic sequence algorithm produces a graph, then the graph it produces is guaranteed to have the right degree sequence.

The second example is iffier. A critic might complain that what we're doing is similar to taking a crossword, writing in a few random words without looking at the clues, then pointing to one of the crossing entries and saying, "There's no word with the letters Q, X, blank, O. So there must be a mistake in the crossword." Sure, there could be a mistake. But maybe we just made the wrong guesses?

For the specific degree sequence $5, 5, 5, 2, 2, 1, 1, 1$, we could backtrack and see that if we had made different choices, we'd be equally stuck. However, we don't want a backtracking algorithm; for a longer degree sequence, that would be too tedious. What we want is an argument that justifies the specific choices our algorithm makes.

But what is it that we want? We don't need to prove that a graph with such a degree sequence must have the edges that we guessed. There could be multiple solutions, after all! What we need to prove is that whenever solutions exist, our guess doesn't eliminate them all: that at least one of the solutions has the structure that the graphic sequence algorithm expects.

6.4 The Havel–Hakimi theorem

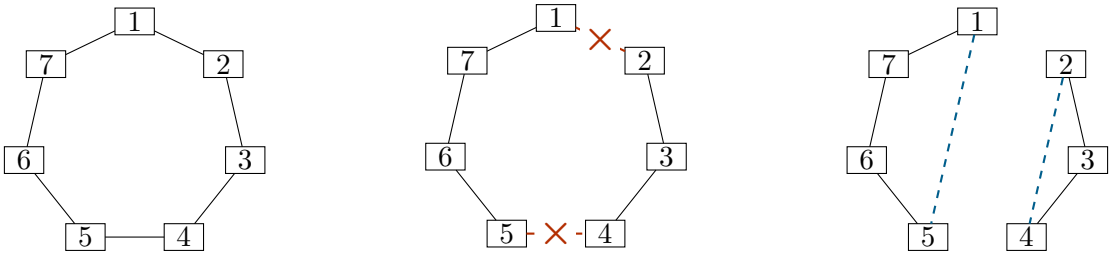
The result we need to prove is known as the Havel–Hakimi theorem: named after Václav Havel, who proved it in 1955 [52], and S. L. Hakimi, who rediscovered the corresponding algorithm for testing if a sequence is graphic in 1962 [46]. Before we prove it, we will restate it in a different way, closer to how it was presented by Havel.

Theorem 6.2 (Havel–Hakimi theorem)**.** *Let G be a graph with vertex set $V(G) = \{1, 2, \ldots, n\}$ such that $\deg(1) \geq \deg(2) \geq \cdots \geq \deg(n)$. Let $k = \deg(1)$ and let $S = \{2, 3, \ldots, k+1\}$.*

Then there is a graph H with $V(G) = V(H)$ and with $\deg_G(x) = \deg_H(x)$ for all x such that the neighbors of vertex 1 in H are precisely the vertices of S.

Question:	How does the Havel–Hakimi theorem justify the correctness of our algorithm?
Answer:	In the first step, vertex 1 (up to tiebreaks) is exactly the vertex we will process, and the edges from 1 to S are exactly the edges we add. The Havel–Hakimi theorem tells us that if there are any graphs G with the degree sequence we want, then in particular there is a graph H in which the choices we made are correct.

Question:	That only applies to the first step—what about future steps?
Answer:	In our algorithm, there are never any edges between two active vertices. So if we only consider the subgraph induced by the active vertices (with the leftover demands), then every step is exactly like the first.



(a) The initial graph (b) Swap edges 12 and 54... (c) ...for edges 15 and 24.

Figure 6.4: An example of an edge swap

The advantage of stating the Havel–Hakimi theorem the way we did is that it describes a transformation: we want to take graph G and transform it into another graph H, without changing the vertices or their degrees, so that H has the edges we want. We'll do the transformation step by step, so we need a way to make a minor change to a graph without changing its degree sequence.

This minor change is an operation called an *edge swap*. To perform one in a graph G, we first find four vertices vw and xy such that vw and xy are edges of G, but vx and wy are not. Then, we delete the edges vw and xy that did exist, and replace them by the edges vx and wy that previously did not exist. Figure 6.4 shows an example of an edge swap. You can see how the affected vertices v, w, x, and y each lose an edge and gain another edge; thus, the overall degree sequence does not change.

We begin by proving a lemma that essentially says, "we can always use edge swaps to make progress toward what we want." (I have sneakily reduced the number of hypotheses, eliminating ones we don't really need. vertex x in Lemma 6.3 is vertex 1 in the Havel–Hakimi theorem, which has the highest degree in G, but we will not to assume this.)

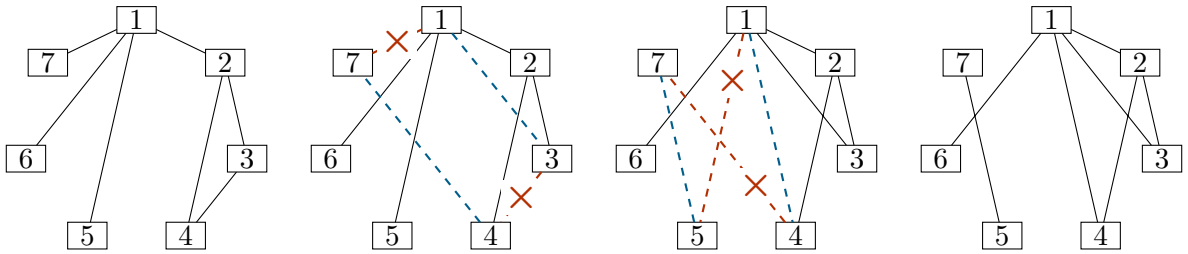
Lemma 6.3. *Let x be a vertex in a graph G and let S be a set of at most $\deg(x)$ vertices in G not equal to x with the highest degrees, breaking ties arbitrarily.*

If not all vertices of S are adjacent to x, then we can perform an edge swap in G to increase the number of vertices in S adjacent to x.

Proof. Let $y \in S$ be a vertex not adjacent to x. Since $\deg(x) \geq |S|$, but x is not adjacent to every vertex in S, it must also have neighbors outside S; call one such neighbor z. To summarize, xy is not an edge of G, but we wish it were; xz is an edge of G, but we don't want it.

We want to fix this state of affairs with an edge swap, but that needs a fourth vertex.

Question:	What kind of fourth vertex w lets us perform an edge swap that delete edge xz and add edge xy?
Answer:	At the same time, we must also delete edge yw and add edge zw. So w must be adjacent to y, but not to z.



(a) x has one edge to S (b) First edge swap (c) Second edge swap (d) Final result

Figure 6.5: Two iterations of Lemma 6.3 in an example; $x = 1$ and $S = \{2, 3, 4\}$

(It does not matter whether $w \in S$ or not, and it does not matter whether w is adjacent to x or not.)

This is where our choice of S becomes important. We chose the vertices of S to have the highest degrees. Even if we broke some ties to do it, we know that the degree of every vertex in S is greater than or equal to the degree of every vertex not in S, other than x. In particular, $\deg(y) \geq \deg(z)$.

On the other hand, we already have an example of a vertex adjacent to z, but not y: vertex x. Before we look at any other vertices, z is in the lead. To reach $\deg(y) \geq \deg(z)$, y must counter that lead at some point: it must have a neighbor z does not. Such a neighbor is exactly the vertex w we wanted.

By performing the edge swap which deletes edges xz and yw and adds edges xy and zw, We achieve the result we want: x is adjacent to one more vertex of S. □

Figure 6.5 shows two examples of using the edge swap given by Lemma 6.3. Throughout, we want vertex $x = 1$ to become adjacent to all three vertices of $S = \{2, 3, 4\}$. (In fact, $\deg(x) = 4$, which is bigger than $|S|$. In the proof of the Havel–Hakimi theorem, we apply Lemma 6.3 to a case where $\deg(x) = |S|$, but only the inequality $\deg(x) \geq |S|$ is necessary.)

1. In Figure 6.5a, x has only one neighbor in S: vertex 2. We take $y = 3$ (a neighbor x wants to have in S, but doesn't) and $z = 7$ (a neighbor x has, but doesn't want). We look for a third vertex adjacent to y, but not z, and find vertex $w = 4$ (vertex 2 would also have worked). Then, we swap edges 17 and 34 for edges 13 and 47, as shown in Figure 6.5b.
2. After we do this, x only has two neighbors in S: vertices 2 and 3. It is still not adjacent to vertex 4, so we take $y = 4$; we take $z = 5$ to be the neighbor that x is going to lose. The two vertices adjacent to $y = 4$ but not $z = 5$ are 2 and 7; let's take $w = 7$. Then, we swap edges 15 and 47 for edges 14 and 57, as shown in Figure 6.5c.
3. In the final result, shown in Figure 6.5d, x is adjacent to every vertex of S.

I put Lemma 6.3 first because from here, the proof of the Havel–Hakimi theorem is essentially just an application of the lemma, together with the extremal principle. It is a good exercise to try to do this yourself before reading ahead.

Proof of Theorem 6.2. Of all graphs with the same vertex set and vertex degrees as G, let H be chosen to maximize the number of neighbors of 1 in S.

Then actually, all vertices of S must be adjacent to 1. If not, we could use Lemma 6.3 (taking $x = 1$) to increase the number of neighbors of 1 in S. But H was chosen to have the largest possible number of such neighbors, so this can't happen.

Therefore H is exactly the graph needed to prove the theorem. □

I should say that the graphic sequence algorithm we've described is also known as the Havel–Hakimi algorithm, but it is commonly described in a different way that I consider less intuitive. Instead of putting the demands on the vertices and drawing edges, we can think of the algorithm as modifying a sequence $d_1 \geq d_2 \geq \cdots \geq d_n$ of the intended degrees. If $k = d_1$, the highest degree, then the algorithm places edges from the first vertex to the k vertices after it. This corresponds to an operation on sequences, turning $d_1, d_2, \ldots, d_n$ into

$$\underbrace{d_2 - 1, d_3 - 1, \ldots, d_{k+1} - 1}_{k \text{ terms}}, d_{k+2}, \ldots, d_n.$$

Repeating this operation eventually produces a sequence of all 0's (in which case the original sequence is graphic) or a sequence which is clearly not graphic (for example, because it contains negative numbers).

6.5 More on degree sequences

We only used edge swaps in the proof of the Havel–Hakimi theorem, but they have other applications. To begin with, we can show the following theorem:

Theorem 6.4. *If two graphs G and H have the same vertex set V, and $\deg_G(x) = \deg_H(x)$ for all $x \in V$, then we can turn G into H by doing edge swaps.*

Proof. We prove this by induction on the size of V (the common vertex set of G and H).

When $|V| = 1$, there is nothing to show: there is only one possible graph on 1 vertex.

Assume this is possible for all pairs of $(n-1)$-vertex graphs, and let G and H both have n vertices. Let x be a vertex with the highest degree (in both G and H) and let S be the $\deg(x)$ vertices with the highest degrees after x.

By applying Lemma 6.3 repeatedly, we can perform edge swaps on G to get a graph G' in which x's neighbors are the vertices in S. We can do the same to H, getting a graph H' (and so there is also a sequence of edge swaps that turn H' into H, by reversing those edge swaps).

By induction, there is a sequence of edge swaps that turns $G' - x$ into $H' - x$ (both $(n-1)$-vertex graphs on the same vertex set). We can perform these edge swaps on G' instead, and they will work equally well, turning G' into H'. Finally, we know that we can turn H' into H.

This shows that we can turn G into H, and so by induction, this works when G and H have any number of vertices. □

Why is this theorem useful? Well, we know how to answer two possible questions about potential degree sequences...

1. Is this sequence graphic? (Is it the degree sequence of a graph?)

2. How is this sequence graphic? (Find a graph with this degree sequence.)

. . . but there is a third, harder question we can ask:

3. What is a typical graph with this degree sequence?

That's deliberately vague. However, Theorem 6.4 is the beginning of a possible answer to the last question.

What we can do is start with an arbitrary graph which has that degree sequence, then perform many edge swaps at random, getting a random graph with this degree sequence.[11] We can approximately answer questions like "are most graphs with this degree sequence connected" or "what is the average diameter of a graph with this degree sequence" by sampling many random graphs and analyzing the set of results.

We can even approximately answer questions like "how many different graphs have this degree sequence?" To do this, just sample lots of graphs, then count how many repeats you get. Compare this to how many repeats you'd expect, if there were N possible graphs total. This should let you estimate the most likely value of N.

There is one difficulty to the procedure I'm outlining to you so cheerfully and optimistically, and that is the requirement of performing "many" edge swaps. Obviously, just one or two random edge swaps are not enough: the result is guaranteed to be very similar to our non-random starting graph. As the number of edge swaps grows, the dependence on the starting graph slowly decreases. However, it's still not known in general how many edge swaps are necessary to make that dependence negligible. Without that knowledge, the random sampling algorithm cannot be considered reliable.

6.6 Practice problems

1. Using the degree sequence algorithm, or in some other way, determine which of the sequences below are graphic:

   a) $7, 6, 5, 4, 3, 2, 1$.

   b) $4, 4, 3, 3, 3, 2, 2$.

   c) $6, 6, 5, 3, 2, 2, 1, 1$.

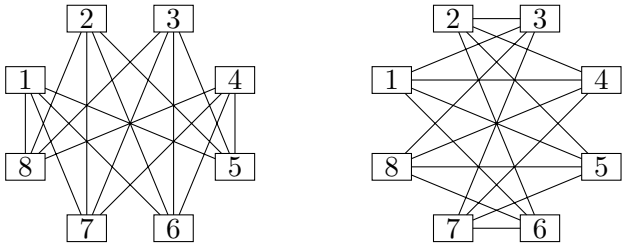
   d) $n, n, n, \underbrace{3, 3, \ldots, 3}_{n \text{ times}}$ (for any n).

   e) $n-1, n-1, n-1, n-1, \underbrace{3, 3, \ldots, 3}_{n-4 \text{ times}}$ (for any n).

   For the sequences that are graphic, construct graphs with those degree sequences.

[11] Fine print: this does not sample a graph uniformly at random. Essentially, it samples a graph with probability proportional to how many edge swaps are possible to do in it. But that's a known bias we can account for.

2. Let G be the graph on the left in the diagram below, and let H be the graph on the right. Corresponding vertices have the same degrees, and the two graphs are isomorphic, but they are not the same graph: they have different edges. G has all edges between $\{1, 2, 3, 4\}$ and $\{5, 6, 7, 8\}$, while H has all edges between $\{1, 2, 5, 6\}$ and $\{3, 4, 7, 8\}$.



   Find a sequence of edge swaps to turn G into H. (Four edge swaps are enough.)

3. Let me tell you another story.

   The other day, I was at another very large party. Once again, each person at the party knew a different number of people, with one exception: I personally knew the same number of people as another guest, named Pasha. In addition, I can promise you that everyone at this party knew at least one other person.

   The surprising thing is this: I did not know Pasha! How could this happen, and why does this not contradict Proposition 6.1?

4. In Problem 6.1, I included the condition that everyone at the party knew at least one other person. If this condition is dropped, then there is another possible graph: how can we find it? (What should its degree sequence be?)

5. There are 7 possible graphs with the vertex set $\{1, 2, 3, 4, 5\}$ such that $\deg(1) = \deg(2) = 3$ and $\deg(3) = \deg(4) = \deg(5) = 2$. Find all of them, and demonstrate directly that we can turn any one of them into any other via edge swaps.

6. There is another proof of Theorem 6.4 that is closer to our argument for the Havel–Hakimi theorem.

   Specifically, if G and H have the same vertex set, and every vertex has the same degree in G and in H, then we can measure "how far away" G and H are from each other by asking: how many edges does G have that are not also edges of H? Now we can do the following:

   a) Prove that if G and H are not literally the same graph, then there is an edge swap we can perform on G to reduce "how far away" it is from H.

   b) Prove Theorem 6.4 using part (a).

7. Under what conditions on a, b, and n is a sequence of the form

$$\underbrace{a, a, \ldots, a}_{n}, \underbrace{b, b, \ldots, b}_{n}$$

   a graphic sequence?

8. Let G be an n-vertex graph whose vertices $1, 2, \ldots, n$ have degrees $d_1, d_2, \ldots, d_n$ respectively, sorted in descending order:

$$d_1 \geq d_2 \geq \cdots \geq d_n.$$

Then d_1 (the maximum degree of G) can be at most $n-1$; in fact, it can be at most the number of nonzero degrees among $d_2, \ldots, d_n$.

a) Explain why this is equivalent to the following inequality:

$$d_1 \leq \sum_{i=2}^{n} \min\{d_i, 1\}.$$

Here, $\min\{a, b\}$ is simply the smaller of the two numbers a and b.

b) Prove the following inequality, which is a 2-vertex generalization of the previous one:

$$d_1 + d_2 \leq 2 + \sum_{i=3}^{n} \min\{d_i, 2\}.$$

(Consider the number of edges between vertices 1 and 2 and the rest of the graph. Why is the "2+" there?)

c) In general, we have an inequality of the form

$$\sum_{i=1}^{k} d_i \leq f(k) + \sum_{j=k+1}^{n} \min\{d_j, k\}$$

for each $k \in \{1, \ldots, n\}$. What is the best possible function $f(k)$ that makes this inequality true? (I ask for "best possible" because some incredibly big function like $f(k) = 2^{2^{2^k}}$ will also work, but will not be as useful as the smallest possible number you could put there.)

More is true: a theorem of Pál Erdős and Tibor Gallai [30] states that if a sequence $d_1, d_2, \ldots, d_n$ satisfies the inequality in part (c) for every k, and the sum $d_1 + d_2 + \cdots + d_n$ is even, then the sequence is graphic. This is an alternative way to check whether a sequence is graphic.

7 Multigraphs and digraphs

The purpose of this chapter

The first half of this chapter introduces multigraphs: a more general version of graphs, allowing loops and duplicate (parallel) edges. By putting this content here, I've made my choice between two unsatisfying options: should multigraphs be an exception, or should they be the default?

I've decided they should be the exception, not introduced until Chapter 7, and not considered to be graphs except in a few asides in later chapters. The advantage of this option is that it simplifies some definitions, especially early on when you have enough new complexity to worry about; also, in many applications, it is all you need from graph theory.

The option I didn't take would have been to make Definition 7.1 the definition of a graph, and to refer to graphs without loops and parallel edges as simple graphs. (I will still use this term when I need to emphasize that I'm not considering a multigraph.) The advantage of this option is that it lets you state and prove every result in fuller generality.

In practice, the differences between simple graphs and multigraphs almost always fall somewhere on the following spectrum:

1. They apply to multigraphs only: even when you start with a simple graph, you will be forced to handle multigraphs to proceed.
2. They apply to both simple graphs and multigraphs, and there are applications where the multigraph version is useful.
3. They apply to both simple graphs and multigraphs, but not in a way that's useful: maybe the loops and parallel edges can always be ignored, or maybe their existence trivializes the problem.
4. They apply only to simple graphs.

In future chapters, I will clearly call out situations of the first two types, and try to mention the other situations in passing if it seems unclear.

When it comes to directed graphs, introduced in the second half of the chapter, the situation is a bit different. Although some of the theory is the same as for undirected graphs, by default, results don't carry over. Accordingly, I will discuss directed graphs in a few big chunks, outside this chapter: all of Chapter 12 and Chapter 28 deal with directed graphs, and one section is devoted to them in Chapter 8 and in Chapter 20.

Notably, the problem of finding Euler tours discussed in Chapter 8 is important to solve for both multigraphs and for directed graphs, and I think it's the first such problem in the textbook; this is one of the reasons I've placed this chapter right before it.

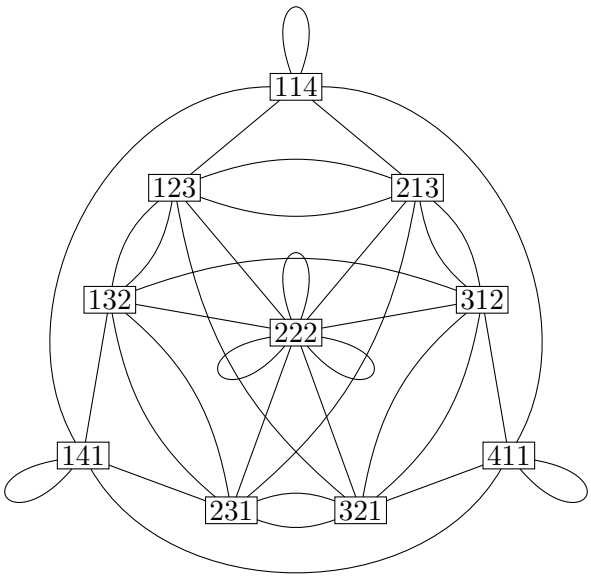


Figure 7.1: A multigraph representation of the 6-pebble game

7.1 Multigraphs

Consider a game[12] with the following rules. There are six pebbles, split between three piles; each pile must contain at least one pebble. There are two types of valid moves: you may swap two piles, or you may move a pebble from one pile to another, as long as the starting pile has at least two pebbles in it.

Figure 7.1 shows a graph of the possible states of the 6-pebble game (with vertex xyz representing a state with x, y, and z pebbles in the three piles); edges in the graph represent valid moves. Well... it doesn't quite show a graph of the game: it has some edges that a graph cannot have. These edges represent moves that do not change the state, as well as situations where two different moves accomplish the same result.

Question:	Why would we draw a loop from vertex 114 back to itself?
Answer:	Swapping the first two piles in state 114 just takes us back to state 114, since the two piles have the same number of pebbles.

Question:	Why are there two edges between vertices 123 and 213?
Answer:	The game has two ways to turn 123 into 213 (or the reverse): we can move a pebble from the 2-pebble pile to the 1-pebble pile, or we can swap the 1-pebble pile with the 2-pebble pile.

[12] It's a "game" because it has rules, but there is no victory condition. Sorry.

Question:	Okay, so what's going on at vertex 222?
Answer:	The six "reasonable" edges out of vertex 222, going to vertices 123, 132, 213, 231, 312, and 321, all represent moving a pebble from one pile to another. There are also three moves that accomplish nothing: we can pick any two piles and swap them.

We cannot have these edges in a graph, so we define a multigraph to be a more general kind of structure that allows these edges to exist. The formal definition needs to be a bit more complicated, though.

Definition 7.1. *A **multigraph** G is a triple consisting of an arbitrary set of **vertices** $V(G)$, an arbitrary set of **edges** $E(G)$, and a symmetric **incidence relation** between them: we say that vertex x is **incident** to edge e, or e is incident to x, if they are related by the incidence relation.*

*In a multigraph, two vertices are called **adjacent** if there is an edge incident to both.*

Previously, we defined edges as pairs of vertices. This is no longer possible, because two edges can have the same pair of endpoints; to distinguish them, there must be something more to their identity. In diagrams (such as in Figure 7.1) we often don't mark the difference between such edges, but to deal with them formally, we would need to give them individual labels.

We have special names for the two situations that can exist in a multigraph, but not in a simple graph:

Definition 7.2. *An edge in a multigraph which has only one endpoint is called a **loop**. Two edges are called **parallel** if they have the same set of endpoints.*

For example, in Figure 7.1, there is a loop incident to vertex 114, two parallel edges joining vertices 123 and 213, and three parallel loops incident to vertex 222.

There are relationships between graphs and multigraphs in both directions:

Definition 7.3. *A **simple graph** is a graph. The **simplification** of a multigraph G is the simple graph with the same vertex set, and an edge $\{x, y\}$ (for $x, y \in V(G)$ with $x \neq y$) whenever G has at least one edge incident to both x and y. Effectively, this removes all loops, and replaces parallel edges with a single edge.*

When a multigraph has no loops or parallel edges, we will treat it as the same object as its simplification; even though on a low level they are defined differently, reasoning about them in practice yields identical outcomes.

Many notions that we've introduced for graphs still make sense for multigraphs, with some modifications. There are two relatively large changes that are worth spending some time discussing: one related to walks, paths, and cycles, and another related to vertex degrees.

A walk in a multigraph is no longer just a sequence of vertices: it matters how we get from one vertex to another. For example, in the 6-pebble game, we consider a multigraph model if it matters which kind of move we used at each step. (If this distinction does not matter, then we do not need the added complexity of multigraphs.) Thus:

Definition 7.4. *An* $x - y$ *walk of length* l *in a multigraph* G *is a sequence*

$$(x_0, e_1, x_1, e_2, x_2, \ldots, x_{l-1}, e_l, x_l)$$

that alternates between vertices and edges of G*, satisfying the following:* $x = x_0$*,* $y = x_l$*, and for each* $i = 1, \ldots, l$*, the endpoints of edge* e_i *are* x_{i-1} *and* x_i*.*

For example, suppose that we start in state 114 of the pebble game, and we make three moves: we swap the first two piles, then move one pebble from the last pile to the first, then swap the first two piles again. To represent this as a walk, we'll in the 6-pebble multigraph, we will first need to invent notation for the different types of edges. Let's say that $s_{ij}\{x, y\}$ denotes a move between states x and y by swapping piles i and j, and $m_{ij}\{x, y\}$, denotes a move between states x and y by moving a pebble between piles i and j. (Keep in mind that edges don't have a distinguished start and end, so $s_{ij}\{x, y\}$ is the same edge as $s_{ij}\{y, x\}$.) Then our walk would be represented by the sequence

$$(114,\ s_{12}\{114, 114\},\ 114,\ m_{31}\{114, 213\},\ 213,\ s_{12}\{213, 123\},\ 123).$$

This is very cumbersome, so usually we don't pull out the full notation unless we need it. Even if we're not looking this closely, though, the definition still matters. For example, when listing the different walks between two vertices, it is important to know when two walks are considered to be different: they are different if the two sequences are different.

Paths and cycles have the same definition, but with new implications. Previously, we were not able to define cycle graphs C_1 or C_2; this is possible in the world of multigraphs. The multigraph C_1 is a single vertex with a loop; the multigraph C_2 has two vertices and two edges between them.

In all cases, to pick out a path and cycle in a multigraph, we need to specify not only the vertices and the order in which they occur, but also the specific edges we use between those vertices. The upshot is that paths and cycles can still be represented by walks (closed walks, in the case of a cycle), but they must be the multigraph walks defined in Definition 7.4.

If there's any bright side to this, it is that it's easier to tell when a closed walk represents a cycle. This is true of a closed walk

$$(x_0, e_1, x_1, e_2, x_2, \ldots, x_{l-1}, e_l, x_0)$$

as long as $e_1, e_2 \ldots, e_l$ are distinct, vertices $x_0, x_1, \ldots, x_{l-1}$ are distinct, and $l \geq 1$.

7.2 Degrees in multigraphs

We would like to generalize the definition of vertex degrees to apply to multigraphs, while giving the same answer as our old definition for simple graphs when there happen to be no loops or parallel edges. It is not obvious how to do so in a natural way, so let's stop and think for a moment about it.

Definitions are not set in stone: as mathematicians, we can choose how to make them. However, we should try to make interesting and useful definitions. Pay attention to the reasoning here if you want to understand why definitions are the way they are!

To begin with, for simple graphs, we can count the degree of a vertex x in two equivalent ways: by counting the edges incident to x, or by counting the vertices adjacent to x. In a multigraph, these two methods can give different answers: for example, in Figure 7.1, vertex 123 is incident to 7 edges, but only has 5 neighbors!

We must immediately reject the second method of counting, because it completely ignores the multigraph structure: why bother having two edges between 123 and 132 if it's not going to affect the degree of either vertex? In other words, counting vertices adjacent to x yields the degree in the simplification of the multigraph.

The first method of counting is not bad, and could have ended up the definition. However, it also has a problem.

By this rule, the degree sequence of the 6-pebble graph would be $9, 7, 7, 7, 7, 7, 7, 5, 5, 5$. By the handshake lemma (Lemma 4.1), which we proved in Chapter 4 for simple graphs, the sum of degrees in a graph is twice the number of edges; applying that here, we would conclude that the graph has $\frac{9+6\cdot 7+3\cdot 5}{2} = 33$ edges. But this is incorrect: if you count the edges in Figure 7.1 one by one, you will get 36.

Question:	Where does the discrepancy between 33 and 36 come from?
Answer:	It comes from loops: when we naively apply the handshake lemma, the 6 loops in the graph only get counted as 3.

Question:	Looking at the proof of the handshake lemma, why does this happen?
Answer:	In the proof, we assumed that adding an edge to a graph increases the degree sum by 2. However, if the degree of a vertex were defined to be the number of incident edges, then a loop would only increase the degree of a single vertex by 1.

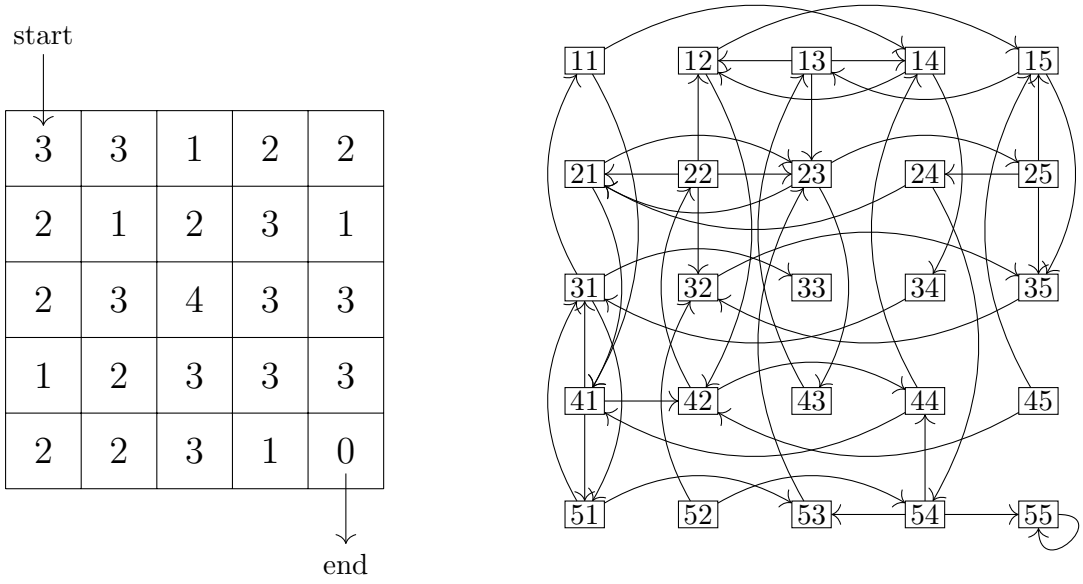
The handshake lemma is so valuable that we are willing to add a special case to our definition, just for this purpose. To rescue the lemma, we need the degree sum to increase by 2 when a loop is added to a graph. It wouldn't make any sense for a loop incident to a vertex x to increase the degree of any vertex other than x; thus, it must contribute 2 to the degree of x. This tells us what the definition of vertex degree must be:

Definition 7.5. *The **degree** of a vertex x in a multigraph G is the number of edges incident to x, counting every loop incident to x twice.*

By this definition, the degree sequence of the 6-pebble graph is $12, 7, 7, 7, 7, 7, 7, 6, 6, 6$.

You might ask: is there a graphic sequence problem for multigraphs? There is, but the answer to it is much less exciting than it is for graphs. Practice problem 3 at the end of this chapter gives the condition, and asks you to discover the proof yourself.

Finally, the definition of a graph isomorphism changes when it is applied to multigraphs. There are two equivalent ways to make that change:



(a) A numerical maze

(b) A digraph representation of the maze

Figure 7.2: Solving a maze using directed graphs

- We could say that an isomorphism between multigraphs G and H is still a bijection $\varphi\colon V(G) \to V(H)$, but with a slightly different condition on φ. Rather than simply preserving adjacency, it should be the case that for any vertices x and y in G (possibly equal) the number of edges between x and y should be the same as the number of edges between $\varphi(x)$ and $\varphi(y)$.
- We could also say that an isomorphism between multigraphs G and H is a pair of bijections: a bijection $\varphi\colon V(G) \to V(H)$, and a second bijection $\varphi'\colon E(G) \to E(H)$. The condition relating these should now be that for a vertex $x \in V(G)$ and an edge $e \in E(G)$, x is incident to e if and only if $\varphi(x)$ is incident to $\varphi'(e)$: the bijections φ and φ' preserve the incidence relation.

These two options are equivalent, and therefore equally good, but the second is more in the spirit of our definition of a multigraph. The intuition you should have is that a discrete structure (like a graph or a multigraph) consists of two parts: some objects, and some relationships between them. In the case of a multigraph, the objects are the vertices and edges, and the relationships between them are given by the incidence relation. An isomorphism between two discrete structures should be a bijection between the objects that preserves the relationships. This idea will guide you to the correct definition of an isomorphism both in graph theory and outside it.

7.3 Directed graphs

Figure 7.2a shows an unusual kind of maze: a numerical maze. Like a normal maze, it has a start (the top left cell) and an end (the bottom right cell). Instead of paths and walls, however,

you solve the maze by using the numbers: if you're in a cell with the number k, then you can jump to another cell that's k steps away either horizontally or vertically.

We might try to represent an ordinary maze by a graph, so that a path in the graph from the start vertex to the end vertex will give us a solution to the maze. In the numerical maze, we encounter an unexpected difficulty: the jumps we make in the maze are often one-way. From the top left corner, which has a 3 in it, we can jump to a cell three steps to the left or to a cell three steps down. However, neither of these numbers contains a 3, so neither of them will let us return to where we just were! As a result, a graph is unsuitable as a model for this maze: adjacency between the cells is an asymmetric relationship.

The structure we use in such cases is called directed graph, or digraph for short. In a diagram, we draw a directed graph by making each edge an arrow, rather than a line, as in Figure 7.2b. An arrow from vertex x to vertex y represents, informally, an adjacency that exists only when going from x to y, not when going from y to x. The formal definition is:

Definition 7.6. *A **directed graph** or **digraph** D is a pair consisting of an arbitrary set of vertices $V(D)$ and a set of **arcs** or **directed edges** $E(D)$. Each arc is an ordered pair (x, y) where $x, y \in V(D)$.*

*We say that arc (x, y) **starts** at x and **ends** at y; it is an arc **from x to y**, or **out of x** and **into y**.*

In principle, we could combine this definition with the definition of a multigraph in the previous section, obtaining the concept of a directed multigraph. I will not do so in this book; juggling all these definitions is complicated enough as it is! However, even without that added flexibility, we allow the arcs (x, y) and (y, x) to coexist in a digraph: after all, these are not the same arc. For example, in Figure 7.2b, arcs $(21, 23)$ and $(23, 21)$ both exist: we can jump in either direction between this pair of cells, because they are 2 steps apart, and both contain a 2. The pair (x, x) is also a valid ordered pair, so it can also be an arc in a directed graph: it represents a loop from x to x.

It is sometimes convenient to forget about the directions of the arcs, and go back to an undirected object.

Definition 7.7. *The **underlying graph** of a digraph D is the multigraph G with vertex set $V(G) = V(D)$ and an edge incident to x and y for each arc $(x, y) \in E(D)$. The digraph D is called an orientation of G.*

In directed graphs, the notion of vertex degrees becomes even more muddled than it was for multigraphs. Properly speaking, we should not describe the degree of a vertex by a single number. Instead, we count the number of arcs into a vertex and the number of arcs out of a vertex separately.

Definition 7.8. *The **indegree** of a vertex x in a digraph D, denoted $\deg_D^-(x)$, is the number of arcs into x.*

*The **outdegree** of a vertex x in a digraph D, denoted $\deg_D^+(x)$, is the number of arcs out of x.*

As with the ordinary notion of degree, we drop the subscript and write $\deg^-(x)$ and $\deg^+(x)$ if it is clear which digraph containing vertex x we mean.

An even more powerful version of the handshake lemma holds for indegrees and outdegrees in directed graphs. Before reading ahead to see how to prove it, see if you can figure out a proof of Lemma 7.1 yourself, by imitating the proof of Lemma 4.1 in Chapter 4.

Lemma 7.1. *In any digraph D, the vertex indegrees add up to the number of arcs, as do the vertex outdegrees:*

$$\sum_{v \in V(D)} \deg_D^-(v) = |E(D)| = \sum_{v \in V(D)} \deg_D^+(v).$$

Proof. We will prove that for any directed graph with m edges, the identity in the lemma holds, and we will do it by induction on m.

The base case is $m = 0$. The lemma holds in this case because the sum of indegrees, the number of arcs, and the sum of outdegrees are all equal to 0: there are no arcs, so no vertex has a nonzero indegree or outdegree.

Assume that the lemma holds for all directed graphs with $m - 1$ arcs. Let D be a directed graph with m arcs, and let (x, y) be any arc of D. We can apply the induction hypothesis to $D - (x, y)$, a digraph with $m - 1$ arcs.

What is the relationship between the indegrees and outdegrees in D and in $D - (x, y)$? There are three cases:

- In D, the outdegree of x is 1 higher than in $D - (x, y)$, because arc (x, y) contributes 1 to that outdegree.
- In D, the indegree of y is 1 higher than in $D - (x, y)$, because arc (x, y) contributes 1 to that indegree.
- $\deg^+_{D-(x,y)}(v) = \deg^+_D(v)$ and $\deg^-_{D-(x,y)}(v) = \deg^-_D(v)$ in all other cases.

Adding up all the changes, we see that

$$\sum_{v \in V(D)} \deg_D^-(v) = 1 + \sum_{v \in V(D)} \deg^- D - (x, y)(v)$$

and

$$\sum_{v \in V(D)} \deg_D^+(v) = 1 + \sum_{v \in V(D)} \deg^+ D - (x, y)(v).$$

By the induction hypothesis, both of the right-hand sides are equal to $1 + (m - 1) = m$, proving the lemma for D: an arbitrary m-arc directed graph. Then, by induction, the lemma holds for directed graphs with any number of arcs. □

Less commonly, in addition to the indegree and outdegree of a vertex, we define the net degree of x to be $\deg^\pm_D(x) = \deg^+(x) - \deg^-(x)$ and the total degree of x to be $\deg_D(x) = \deg^+(x) + \deg^-(x)$.

Question:	How can we interpret the net degree and the total degree of x?
Answer:	The net degree measures how many more arcs at x go out, compared to in; it tells us whether x is a "net exporter" or "net importer" of arcs. The total degree is the degree of x in the underlying graph.

Question:	What do the net degrees add up to in a directed graph?
Answer:	We can write the sum of net degrees as $$\sum_{v \in V(D)} \deg_D^{\pm}(v) = \sum_{v \in V(D)} \deg^+(v) - \sum_{v \in V(D)} \deg^-(v),$$ which simplifies to 0 by Lemma 7.1.

Question:	What do the total degrees add up to?
Answer:	For a similar reason, they add up to $2\|E(D)\|$. This also follows from the handshake lemma applied to the underlying graph of D.

In directed graphs, the notion of isolated vertices (with indegree and outdegree 0) still makes sense, but it is sometimes useful to consider the indegree and outdegree separately. We call a vertex x in a digraph a *source* if $\deg^-(x) = 0$, and a *sink* of $\deg^+(x) = 0$.

Question:	Does the graph in Figure 7.2b contain any sources?
Answer:	Yes: vertices 45 and 52 are sources. (You might be tempted to call 11 a source because it's a starting point, but it's not: it's possible to return to vertex 11 after leaving it.)

Question:	Does it contain any sinks?
Answer:	Yes: vertex 33 is a sink, because the number in the center of the grid is 4, and there are no cells to hop to that are 4 spaces away in any direction. Sinks in such a graph represent "dead ends" in the maze which cannot be left.

7.4 Directed walks, paths, and cycles

Moving on, the other big change with directed graphs—and, often, the reason we study directed graphs to begin with—is the behavior of walks, paths, and cycles. The definition of a walk in a digraph superficially does not seem very different from the definition we saw in Chapter 3 for undirected graphs:

Definition 7.9. *An* $x - y$ ***walk of length*** l *in a digraph* D *is a sequence*

$$(x_0, x_1, x_2, \ldots, x_l)$$

of vertices of D *where* $x = x_0$*,* $y = x_l$*, and for each* $i = 1, \ldots, l$*,* (x_{i-1}, x_i) *is an arc of* D*.*

The difference is that this definition requires all the arcs (x_{i-1}, x_i) to point in the same direction along the directed walk, and this small difference is a dramatic change. While an $x - y$ walk in an undirected graph is essentially the same as an $x - y$ walk except for which way you go, a directed graph might have an $x - y$ walk but no $y - x$ walk.

By analogy with P_n and C_n, there are two families of directed graphs: the *directed path graph* $\overrightarrow{P_n}$ has vertex set $\{1, 2, \ldots, n\}$ with an arc $(i, i+1)$ for $i = 1, 2, \ldots, n-1$, and the *directed cycle graph* $\overrightarrow{C_n}$ is obtained from $\overrightarrow{P_n}$ by adding the arc $(n, 1)$. Both of these are defined for all $n \geq 1$.

When dealing with directed graphs, the terms *path* and *cycle* refer to copies of the directed path graph and the directed cycle graph, respectively; I may use the terms "directed path", "directed cycle", or "directed walk" for emphasis. A directed path or cycle is represented by a walk $(x_0, x_1, x_2, \ldots, x_l)$ if it has vertices $\{x_0, x_1, \ldots, x_l\}$ and arcs $\{(x_0, x_1), \ldots, (x_{l-1}, x_l)\}$. In the case of a cycle, it follows that the walk representing it is closed, with $x_l = x_0$.

To finish this introduction to directed graphs, let's put the new ideas about directed graphs together into a theorem: the directed version of Theorem 4.4 from Chapter 4.

Theorem 7.2. *Every directed graph* D *with no sinks contains a cycle.*

Proof. Let the walk $(x_0, x_1, x_2, \ldots, x_l)$ represent a path in D of the largest possible length. Because D has no sinks, we know in particular that $\deg^+(x_l) > 0$, so there must exist some arc (x_l, y) that starts at x_l.

If $y \notin \{x_0, x_1, x_2, \ldots, x_l\}$, then the walk $(x_0, x_1, x_2, \ldots, x_l, y)$ would represent a longer path, contrary to our assumption. Therefore $y = x_i$ for some i, and the closed walk $(x_i, x_{i+1}, \ldots, x_l, x_i)$ represents the cycle we wanted. □

Question:	Does the same result hold for directed graphs with no sources?
Answer:	Yes: we can write a similar proof, but look for a vertex y with an arc into x_0, rather than out of x_l.

Question:	Does the converse hold? That is, if D has a cycle, is it true that it has no sources or sinks?
Answer:	No, and the digraph in Figure 7.2b is a counterexample. This digraph has two sources and a sink, and yet it still has many cycles: for example, the length-3 cycle represented by $(12, 15, 13, 12)$.

7.5 Practice problems

1. Consider the 6-pebble multigraph in Figure 7.1.

   a) How many $114-411$ walks of length 1 does it have?

   b) What about length 2?

   c) What about length 3?

   d) For practice, write out one of the length-3 walks as a sequence of vertices and edges, using the $s_{ij}\{x, y\}$ and $m_{ij}\{x, y\}$ notation for edges.

2. Consider the multigraph below. How many closed walks of length 10 begin and end at the vertex on the left?

3. Here is why we consider the graphic sequence problem for simple graphs, and not for multigraphs.

   a) Let $d_1 \geq d_2 \geq \cdots \geq d_n$ be a sequence of nonnegative integers. Show that it is the degree sequence of a multigraph if and only if $d_1 + d_2 + \cdots + d_n$ is even.

   b) Let $d_1 \geq d_2 \geq \cdots \geq d_n$ be a sequence of nonnegative integers. Show that it is the degree sequence of a multigraph with no loops if and only if $d_1 + d_2 + \cdots + d_n$ is even and $d_1 \leq d_2 + d_3 + \cdots + d_n$.

4. Let G be a simple graph with n vertices and maximum degree r. We assume that rn is even, so that an r-regular graph on n vertices exists. (But G might not be r-regular; some of its vertices can have degree less than r.)

   a) Prove that there is an r-regular multigraph H, with $V(H) = V(G)$, such that G is a subgraph of H. (In other words: we can add edges to G to make it regular.)

   b) Give an example of a graph G for which the graph H in part (a) cannot possibly be a simple graph.

5. Solve the numerical maze in Figure 7.2a. What kind of graph-theoretic object represents your solution?

   For an extra challenge, use breadth-first-search, as described in Chapter 3, and find the shortest path through the maze.

6. A game is played with two piles of stones. On a turn, a player picks a pile which is not empty, and takes one or more stones from it. The game ends once both piles are empty.

   We can represent this game by a digraph whose vertices are the states, with arcs representing the valid moves. Let $D_{n,m}$ be the digraph we get when the game is played with a pile of size n and a pile of size m.

   a) Draw a diagram of $D_{3,2}$. (It should have 12 vertices.)

   b) Which vertices of $D_{n,m}$ are sources? Which are sinks?

   c) Are there any values of n and m for which $D_{n,m}$ contains a cycle?

7. The game RPS-7 [71], invented by David C. Lovelace, is an extension of Rock-Paper-Scissors to 7 gestures: Rock, Paper, Scissors, Air, Fire, Sponge, and Water. The gestures have the following relationships:

   - Rock pounds out Fire, crushes Scissors, and crushes Sponge.
   - Fire melts Scissors, burns Paper, and burns Sponge.
   - Scissors swish through Air, cut Paper, and cut Sponge.
   - Sponge soaks paper, uses Air pockets, and absorbs Water.
   - Paper fans Air, covers Rock, and floats on Water.
   - Air blows out Fire, erodes Rock, and evaporates Water.
   - Water erodes Rock, puts out Fire, and rusts Scissors.

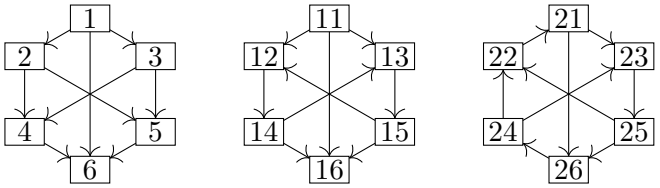
   Let RPS_7 be the graph whose vertices are the 7 gestures, with an arc (x, y) whenever gesture x beats gesture y.

   a) Find a cycle of length k in RPS_7 for each $k = 3, 4, 5, 6, 7$.

   b) For which n can an RPS-n game be constructed so that each object beats the same number of other objects?

8. a) Write a reasonable definition of an isomorphism between two directed graphs.

   b) Here are three directed graphs with isomorphic underlying graphs.



   Prove that as directed graphs, none of these are isomorphic.

8 Euler tours

The purpose of this chapter

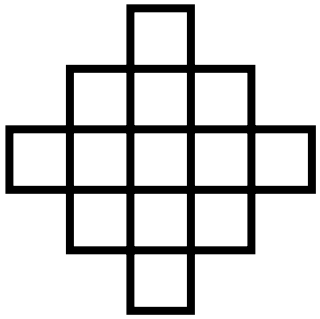
This chapter concludes the part of this book devoted to vertex degrees. It is an application of the concept of vertex degrees, as well as some results from Chapter 4, to solve a problem that could be said to be the beginning of graph theory.

There is a striking contrast in graph theory between the theory of Euler tours and Eulerian graphs, discussed in this chapter, and the theorem of Hamilton cycles and Hamiltonian graphs, discussed in Chapter 17. Superficially, the two problems are very similar, yet we are able to pin down the exact circumstances in which an Euler tour exists; for Hamiltonian cycles, hard work is required in most cases.

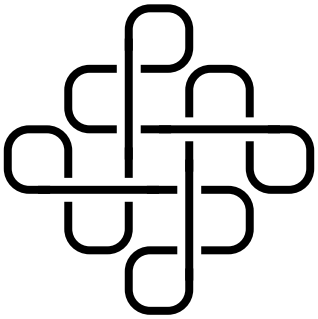
Everything we say about the Euler tour problem for graphs remains true for multigraphs, with no substantial change. For directed graphs, the situation is a bit more complicated, and it is discussed at the end of the chapter. However, many of the important applications of Euler tours, such as the one in Chapter 20, are applications to directed graphs, so this is an important case of the problem to understand.

8.1 Don't lift your pencil!

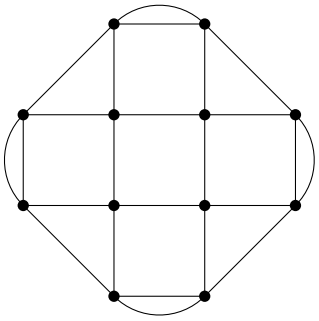
You might have seen a brainteaser like this before! Look at the diagram in Figure 8.1a. Can you draw this shape on paper without lifting your pencil (or pen), and without going over the same line more than once?



(a) Can you draw this figure without lifting your pencil?

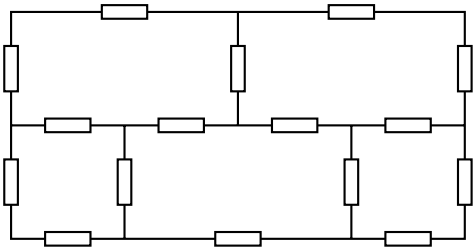


(b) Solution to the challenge in Figure 8.1a

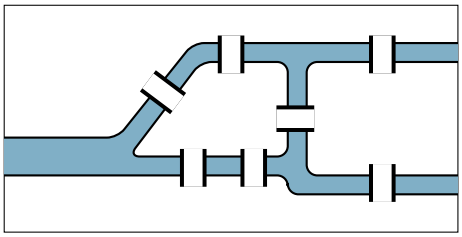


(c) Modeling the puzzle as a multigraph

Figure 8.1: Drawing without lifting your pencil



(a) The five rooms puzzle



(b) The bridges of Königsberg

Figure 8.2: Two more puzzles equivalent to pencil drawing

You can: Figure 8.1b shows you one particularly symmetric way to do it. (To make it clear what the solution is, the lines are bent away from each other slightly to indicate where the path does or does not cross itself.) As a bonus, the path in Figure 8.1b returns to where it started.

This is a book on graph theory, though, not on pencil drawings. So what does this question have to do with graph theory? This is the same question that this book started with: how do we model this problem as a graph?

The places in the drawing that matter most are the places where several lines meet: when tracing the drawing with your pencil, those are the only places where we make a decision. So it makes sense to have a vertex for each of those points. The rest of the drawing consists of lines that go from one such point to another, and their shape does not matter for solving the puzzle: only the starting and ending point matters. So for every such line, we will add an edge between the two vertices it joins. If we apply idea to Figure 8.1a, the result is shown in Figure 8.1c.

Question:	The result is a multigraph; how did that happen?
Answer:	We get parallel edges whenever there are two or more lines that start and end at the same pair of intersection points.

Question:	What sort of graph-theoretic object are we looking for when we solve the brainteaser?
Answer:	We are looking for a walk in the graph which uses every edge exactly once.

There are two other well-known puzzles that are an instance of the same type of problem. One is the five rooms puzzle, shown in Figure 8.2a. Here, five rooms are connected by a total of 16 doors, and the challenge is to plot a path through the rooms that passes exactly once through every door.

Question:	What should the vertices and edges be in the multigraph that makes the five rooms puzzle equivalent to the pencil-drawing puzzle?
Answer:	There should be six vertices: one for each room, where we consider the outside region to be a room as well. Each door between two rooms should be an edge incident to both rooms.

Another such puzzle, the problem of the seven bridges of Königsberg, has historical significance. It was first studied by Leonhard Euler, in 1736 [6], and it may be one of the first mathematical problems to be abstracted into a graph-theoretic model before being solved. (It predates modern graph theory, so the terminology used by Euler was quite different from what you are reading here.) The problem itself is based on a map of the city of Königsberg, shown somewhat abstractly in Figure 8.2b. In Euler's time, there were seven bridges connecting the two sides of the river and the two large islands in the middle of the river. The challenge was to find a path through the city that crossed each bridge exactly once.

Question:	What should the vertices and edges be in the multigraph that models the bridges of Königsberg?
Answer:	There should be a vertex for each of the four landmasses, with an edge for every bridge.

The problem we are solving in all three of these puzzles is named in Euler's honor.

Definition 8.1. *A walk that contains every edge of a graph or multigraph exactly once is called an* ***Euler walk****. If, additionally, the Euler walk is closed, it is called an* ***Euler tour****.*

All three puzzles we have looked at are about finding an Euler walk. Even though the solution in Figure 8.1b happens to be an Euler tour, we have not had a reason to consider Euler tours for their own sake so far. However, you are about to see in this chapter that Euler tours are actually the more fundamental and more natural object. This is reflected in the following definition:

Definition 8.2. *A graph or multigraph is called* ***Eulerian*** *if it has an Euler tour.*

8.2 First steps

The key to determining whether an Euler walk or Euler tour exists turns out to depend on the degree of the vertices. To simplify terminology just for this chapter, we will call a vertex *odd* or *even* if it has odd degree or even degree, respectively.

The following result does not completely answer our question, but is easier to prove than the main theorem of this chapter, so it is where we will begin. I will give the proof in the language of multigraphs, so that we can be sure it works even then.

Lemma 8.1. *In an Eulerian multigraph, every vertex must be even.*

Proof. Let G be an Eulerian multigraph, and let the walk

$$(x_0, e_1, x_1, e_2, x_2, \ldots, e_m, x_m)$$

with $x_m = x_0$ be an Euler tour of G. We prove a stronger statement: if a vertex x appears k times among $x_1, \ldots, x_m$, then $\deg_G(x) = 2k$. Intuitively, the argument for this is that if we

follow the Euler tour around G, then every time the tour enters and leaves x, it uses 2 more edges incident to x.

Formally, let I be the set $\{i : 1 \leq i \leq m \text{ and } x_i = x\}$: the set of all positive positions in the Euler tour where x appears. If $|I| = k$, then the k edges $\{e_i : i \in I\}$ and the k edges $\{e_{i+1} : i \in I\}$ are all the edges incident to x, contributing $2k$ to $\deg_G(x)$. This has two caveats:

- If $x = x_i = x_{i+1}$ for some i, then edge e_i is double-counted in the argument above. However, in this case, edge e_i is a loop with both endpoints at x, contributing 2 to $\deg_G(x)$: it should be double-counted.
- If $x = x_m$, then there is no edge e_{m+1} to count. However, in this case, $x = x_0$ as well, so edge e_1 should be counted instead.

So in both unusual cases, the total contribution to the degree of x is still $2k$.

Every edge of G appears in the Euler tour exactly once, and so by counting the contribution of edges $e_1, \ldots, e_m$ to the degree of x, we compute the degree completely. This shows that $\deg_G(x) = 2k$, an even number; since x was arbitrary, all vertices must be even. □

Question:	Why doesn't Lemma 8.1 solve the problem completely?
Answer:	If every vertex in a multigraph is even, the lemma doesn't guarantee that the multigraph really does have an Euler tour.

In other words, we have shown that the condition "every vertex in G is even" is a necessary condition for G to be Eulerian: that is, G can't be Eulerian without it. We have not determined whether it is a sufficient condition for G to be Eulerian: whether all graphs that satisfy the condition do in fact have Euler tours.

Question:	If $(x_0, e_1, x_1, e_2, x_2, \ldots, e_m, x_m)$ were an Euler walk, but not an Euler tour, what would change in the proof of Lemma 8.1?
Answer:	When $x = x_m$, it would no longer be true that $x = x_0$ as well, so the degree of x would be $2k - 1$ rather than $2k$: x would be odd. Also, when $x = x_0$, the argument wouldn't "notice" the contribution of edge e_1 to x's degree, because I only contains positive integers, so x would be odd in this cases as well.

Following this reasoning carefully gives us the following claim:

Lemma 8.2. *If a multigraph G has an $x - y$ Euler walk, where $x \neq y$, then vertices x and y must be odd, while all other vertices must be even.*

Question:	Which of the puzzles in Figure 8.2 have solutions?
Answer:	Neither puzzle has a solution. If we convert them to multigraphs, then in both cases there will be four odd vertices. However, Lemma 8.2 implies that a graph with an Euler walk can have at most two odd vertices.

The condition "at most two odd vertices" is a bit more complicated than the condition "no odd vertices", but that's not the real reason that Euler tours are taken to be more fundamental than Euler walks.

Imagine for a moment that we had a method for finding Euler tours, but not Euler walks, and suppose that we were faced with a graph G where only two vertices, x and y, were odd. Well, we could fix this quite easily: we could simply add an edge e with endpoints x and y. In the new graph[13] H, all vertices would be even, so H would have an Euler tour.

In our imagination, we already have a method for finding Euler tours, so we could easily find an Euler tour in H. That Euler tour would have to use the new edge e at some point. Well, if we simply delete edge e from the tour, we could follow the tour starting from x and visit all the other edges, ending at y. This is an Euler walk in G.

Outside our imagination, we don't have any systematic method of finding Euler walks or Euler tours, yet. However, this argument lets us reduce the Euler walk problem to the Euler tour problem. It tells us that all we need to focus on is finding Euler tours in graphs where all vertices are even. As soon as we find an algorithm to do this, we will immediately have an algorithm for Euler walks as well!

8.3 Cycle decompositions

Let's take a detour[14] and consider cycle decompositions.

In general, a decomposition of G is a set of subgraphs $\{G_1, G_2, \ldots, G_k\}$ such that every edge of G is an edge of exactly one subgraph G_i. We have already seen one kind of decomposition: the decomposition of a graph into its connected components. We will encounter two other examples of decompositions later: 1-factorizations in Chapter 16 and ear decompositions in Chapter 25.

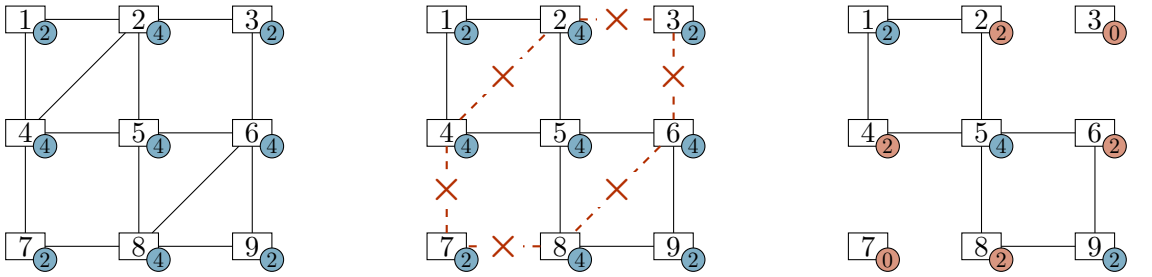
In particular, a *cycle decomposition* of G is a set of cycles in G such that every edge of G is an edge of exactly one of the cycles. This uses the terminology in a straightforward way, but one thing is awkward: the notation. We cannot write a cycle decomposition $\{C_1, C_2, \ldots, C_k\}$, because the notation C_n already refers to the n-vertex cycle graph. When we must, we resort to a standard option all mathematicians take when not allowed to use subscripts: use superscripts instead, writing the cycle decomposition as $\{C^{(1)}, C^{(2)}, \ldots, C^{(k)}\}$.

When studying the Euler tour problem in 1912, Oswald Veblen realized that a single closed walk and a collection of many cycles share a common feature: in both cases, if they visit a vertex k times, they use $2k$ edges incident to that vertex. Therefore the necessary condition for Eulerian graphs is also necessary for having a cycle decomposition. However, Veblen was able to prove [101] that for cycle decompositions, this condition is both necessary and sufficient!

A key step of this proof uses Theorem 4.4: if a graph has minimum degree at least 2, then it contains a cycle. We proved this theorem for simple graphs, but if you go back and examine our argument, you should be able to convince yourself that it can be carried out in multigraphs as well.

[13] Or multigraph: even if G is a simple graph, it might already have an edge xy, so we need to switch to a multigraph model to add edge e.

[14] Heh-heh.



(a) The initial graph G (b) Deleting a cycle C (c) Vertex degrees in $G - C$

Figure 8.3: Finding a cycle decomposition, one cycle at a time

Question:	Alternatively, there is a way to generalize the theorem to multigraphs without looking at the proof of Theorem 4.4 at all. How?
Answer:	If a multigraph has a loop, it has a cycle of length 1, and if it has two parallel edges, it has a cycle of length 2. In both cases, we already get the conclusion we want. If the multigraph has neither of these, then we can think of it as a simple graph.

Theorem 8.3. *A graph has a cycle decomposition if and only if all of its vertices are even.*

Proof. Suppose first that G is a graph with a cycle decomposition $\mathcal{D}$, and let x be an arbitrary vertex of G.

Cycles are 2-regular graphs, so for every cycle $C \in \mathcal{D}$ with $x \in V(C)$, $\deg_C(x) = 2$. Together, the cycles in $\mathcal{D}$ include each edge of G exactly once, so we can split up the degree $\deg_G(x)$ as a sum over the degree of x on cycles. Let $\mathcal{D}_x$ be the set of all cycles in $\mathcal{D}$ containing x: then

$$\deg_G(x) = \sum_{C \in \mathcal{D}_x} \deg_C(x) = \sum_{C \in \mathcal{D}_x} 2 = 2|\mathcal{D}_x|.$$

In particular, $\deg_G(x)$ must be even.

This argument shows that the condition is necessary; now, we must prove that it is sufficient. So let G be a graph in which all vertices are even; our task is to find a cycle decomposition of G. We will do this by strong induction on the number of edges in G. As our base case, if G has 0 edges, then the empty set is a cycle decomposition: it includes every edge exactly once, because there are no edges to include.

Suppose G has $m > 0$ edges. Pick any connected component of G that has at least one edge. In that component, no vertices can have degree 0: an isolated vertex would be a component with no edges, all by itself. Also, no vertices can have degree 1: by assumption, all vertices in G are even. Therefore the component has minimum degree at least 2. By Theorem 4.4, this component contains a cycle C.

In $G - E(C)$ (the graph we get by deleting the edges of C, leaving the vertices as they are), all vertices are still even: the degree of every vertex that lies on C goes down by 2, and all other degrees are unchanged. (An example of such a deletion is illustrated in Figure 8.3.) Also, $G - E(C)$ has fewer than m edges (since C has at least one edge), so our induction hypothesis

applies to it: $G - E(C)$ has a cycle decomposition. Add C to that cycle decomposition, and we get a cycle decomposition of G.

By induction, a cycle decomposition exists for any number of edges. □

This proof also gives us an algorithm for finding a cycle decomposition. Simply find any cycle, set it aside, and repeat with the remainder of the graph until we are out of edges. How do we find a cycle? Well, the proof of Theorem 4.4 begins by taking a longest path, but this is done only to simplify the proof. Really, we can build up a path by aimless wandering through the graph, taking care only not to leave a vertex by the same edge we used to enter it. As soon as we revisit the vertex, we obtain a cycle in the same way that we obtained it from the longest path.

8.4 Gluing cycles together

Before we go on, let me clear the air about one thing: the condition that all vertices are even is not sufficient to guarantee that a graph is Eulerian. However, the exception is rather silly.

Question:	How can you find a graph where all vertices are even, but which does not have an Euler tour? (Small examples include a multigraph with 2 edges and a simple graph with 6 edges.)
Answer:	Just make sure that your graph has multiple connected components with edges in them! A walk can never jump from one component to another, so you can never include all edges of the graph in a single walk.

This is a minor quibble; we will be fine if we restrict our attention to connected graphs. We can even allow graphs with multiple components, as long as all components except for one are merely isolated vertices. The cycle decomposition we get from Theorem 8.3 is the key ingredient in putting together an Euler tour.

Theorem 8.4. *A graph is Eulerian if and only if it is connected (except possibly for isolated vertices) and all of its vertices are even.*

Proof. We know that both parts of this condition are necessary: by Lemma 8.1, all vertices must be even, and we have just discussed why the graph cannot have two components with edges. It remains to show that together, these two conditions are sufficient.

Let G be a graph which has only one connected component aside from isolated vertices, and in which every vertex of G is even. For simplicity, delete all the isolated vertices (which will not affect our argument) so that we can simply assume G is connected. (If G has only isolated vertices, then a walk of length 0 is an Euler tour.) The first thing we'll do is use Theorem 8.3 to find a cycle decomposition $\mathcal{D}$ of G.

Next, we'll build an Euler tour step by step. This is a proof by algorithm. Throughout this process, we will update the following objects:

- a "partial tour" PT, which is a closed walk that uses each edge at most once, but might miss some edges.
- A subgraph H whose edges are exactly the edges missed by PT.
- A set $\mathcal{U}$ of "unprocessed cycles": a cycle decomposition of H.

Initially, we'll take PT to be a walk representing one cycle $C \in \mathcal{D}$. Let $H = G - E(C)$, and let $\mathcal{U} = \mathcal{D} - \{C\}$. We will stop when $\mathcal{U} = \varnothing$.

One by one, we will remove a cycle from $\mathcal{U}$, and splice it into PT. To make this possible, it's important to find a cycle in $\mathcal{U}$ that contains one of the vertices visited by PT.

There must always be such a cycle! Assuming otherwise, let S be the set of vertices visited by PT: S is not empty, but also does not contain any vertices of the cycles in $\mathcal{U}$. Every edge of G is either used by PT (and joins two vertices in S) or is an edge of a cycle in $\mathcal{U}$ (and joins two vertices not in S). No edge of G has exactly one endpoint in S, and by Lemma 3.2, G is not connected, which is a contradiction.

So let's suppose that we have found such a cycle C. Let PT be the closed walk

$$(x_0, x_1, x_2, \ldots, x_{l-1}, x_0)$$

and let $C \in \mathcal{U}$ be a cycle containing a vertex x_i. Choose a closed walk

$$(y_0, y_1, y_2, \ldots, y_{k-1}, y_0)$$

to represent C with the property that $y_0 = x_i$. Then to incorporate C into PT, replace PT by the closed walk

$$(x_0, x_1, x_2, \ldots, x_{i-1}, \underbrace{y_0, y_1, y_2, \ldots, y_{k-1}, y_0}, x_{i+1}, \ldots, x_{l-1}, x_0).$$

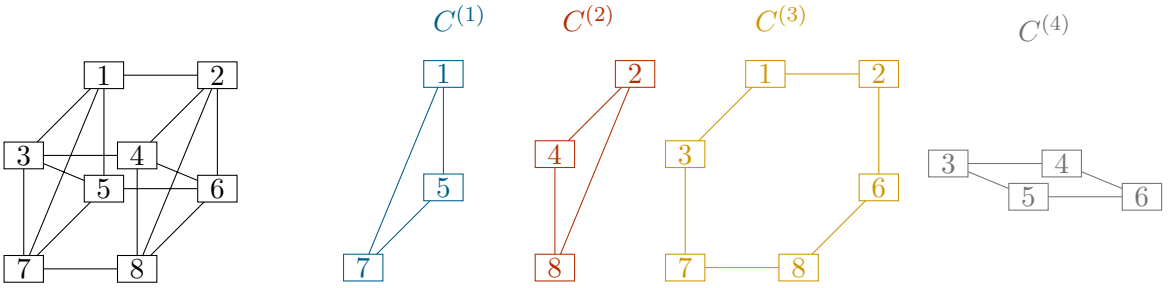
That is, the vertex x_i is replaced by the entire closed walk representing C. Replace H by $H - E(C)$, and $\mathcal{U}$ by $\mathcal{U} - \{C\}$.

After this process, every edge used by the old PT is still used by the new PT, and so is every edge in $E(C)$: after all, $E(C) = \{y_0y_1, y_1y_2, \ldots, y_{k-1}y - 0\}$, and all of these appear as pairs of consecutive vertices in the new PT. Therefore H is still the subgraph whose edges are exactly the edges missed by PT, and $\mathcal{U}$ is still a cycle decomposition of H. Also, it's still true that the new PT uses every edge at most once: all of the new edges came from H, so they did not appear in the old PT.

Repeat this until $\mathcal{U}$ is empty. Throughout the algorithm, it's been true that all edges of G are either part of PT, or part of a cycle in $\mathcal{U}$. Now that there are no cycles in $\mathcal{U}$, all edges must be part of PT, so PT is an Euler tour. □

To truly understand the proof and the algorithm, let's go through an example, finding an Euler tour of the graph in Figure 8.4a. We will begin with a cycle decomposition. The one shown in Figure 8.4b is actually not the best one to use—life would be easier for us if we had fewer longer cycles. The decomposition with many short cycles will let us see more steps of the algorithm in action.

1. Initially, our partial tour PT is a walk $(1, 5, 7, 1)$ representing $C^{(1)}$.



(a) A modified cube graph

(b) A cycle decomposition $\{C^{(1)}, C^{(2)}, C^{(3)}, C^{(4)}\}$

Figure 8.4: A graph and its cycle decomposition, in preparation for finding an Euler tour

2. Next, we want to add a cycle that shares a vertex with PT: that is, we can't use $C^{(2)}$. Let's take $C^{(4)}$ instead; $C^{(3)}$ would also work.

   The only vertex of $C^{(4)}$ appearing on PT is vertex 5. So we choose to represent $C^{(4)}$ by the closed walk $(5, 3, 4, 6, 5)$, and replace the 5 in PT by this closed walk: now,

   $$PT = (1, \underbrace{5, 3, 4, 6, 5}, 7, 1).$$

3. At this point, PT has lots of vertices, so we can add either remaining cycle. Let's add $C^{(2)}$. It shares vertex 4 with PT, so we represent $C^{(2)}$ by $(4, 2, 8, 4)$ and splice it into PT: now,

   $$PT = (1, 5, 3, \underbrace{4, 2, 8, 4}, 6, 5, 7, 1).$$

4. Finally, only $C^{(3)}$ is left to add; it shares many vertices with PT, but in particular, it shares vertex 1, so we choose to represent it by $(1, 2, 6, 8, 7, 3, 1)$. We replace the first occurrence of 1 in PT by this closed walk, getting the final result:

   $$PT = (\underbrace{1, 2, 6, 8, 7, 3, 1}, 5, 3, 4, 2, 8, 4, 6, 5, 7, 1).$$

To check that PT is an Euler tour for yourself, draw all the vertices of the graph in Figure 8.4a, with no edges between them. Then, follow PT, drawing each edge as you use it. As you go, check that you never draw an edge more than once; at the end, check that you have drawn all of Figure 8.4a.

8.5 Variations on Euler tours

Of course, now that we have an algorithm for Euler tours, we immediately have an algorithm for Euler walks, as well as a test for their existence. The only thing that changes is that a graph with two odd vertices can still have an Euler walk.

Question:	In a graph with exactly two odd vertices, is it possible for them to be in different connected components?
Answer:	It is not: if this happened, then looking at one of those components by itself, we'd see a subgraph with only one odd vertex. However, by Corollary 4.2 to the handshake lemma, the number of odd vertices in any graph is even.

Question:	Can you think of a way to modify the algorithm for Euler tours so that it finds an $x - y$ Euler walk instead?
Answer:	Instead of a partial tour PT, begin with a partial walk PW representing an $x - y$ path P, and let $H = G - E(P)$. In H, all vertices are even, so we can take $\mathcal{U}$ to be a cycle decomposition of H. From here, the algorithm can be continued as before.

Another interesting problem is the problem of finding Euler tours in a directed graph. The first novelty in the directed setting is that we no longer check for whether a vertex is even: entering and leaving a vertex has a different effect on its indegree and outdegree.

Question:	If a directed graph D has an Euler tour, what can you say about the degree of a vertex that appears on the tour k times?
Answer:	Each arc into the vertex adds 1 to its indegree, and each arc out of the vertex adds 1 to its outdegree. Thus, the vertex will have indegree k and outdegree k.

Question:	What, then, is the corresponding necessary condition for a digraph to be Eulerian?
Answer:	Every vertex must have an indegree equal to its outdegree. (We can call such vertices *balanced*.)

To prove a necessary and sufficient condition for directed graphs, we must retrace the steps we took in the undirected setting. The only new ingredient is the result we use to find a single cycle; we proved it in Chapter 7. Can you anticipate it before it is used?

Lemma 8.5. *A digraph has a cycle decomposition if and only if all of its vertices are balanced.*

Proof. Suppose first that D is a digraph with a cycle decomposition $\mathcal{D}$, and let x be an arbitrary vertex of D.

In the directed cycle graph $\overrightarrow{C_n}$, every vertex has indegree 1 and outdegree 1. Together, the cycles in $\mathcal{D}$ include each arc of D exactly once, so we can split up the indegree and outdegree

of x as a sum over cycles. Let $\mathcal{D}_x$ be the set of all cycles in $\mathcal{D}$ containing x: then

$$\deg_D^+(x) = \sum_{C \in \mathcal{D}_x} \deg_C^+(x) = \sum_{C \in \mathcal{D}_x} 1 = |\mathcal{D}_x|, \text{ and}$$
$$\deg_D^-(x) = \sum_{C \in \mathcal{D}_x} \deg_C^-(x) = \sum_{C \in \mathcal{D}_x} 1 = |\mathcal{D}_x|.$$

In particular, $\deg_D^+(x) = \deg_D^-(x)$: x is balanced.

This argument shows that the degree condition is necessary; now, we must prove that it is sufficient. So let D be a digraph in which all vertices are balanced; our task is to find a cycle decomposition of D. We will do this by strong induction on the number of arcs in D. As our base case, if D has 0 arcs, then the empty set is a cycle decomposition: it includes every arc exactly once, because there are no arcs to include.

Suppose D has $m > 0$ arcs; the same is true if we pass to D' by deleting all isolated vertices of D. If a balanced vertex has outdegree 0, it also has indegree 0, so it is an isolated vertex; therefore for all $x \in V(D')$, $\deg^+(x) \geq 1$. By Theorem 7.2, D' contains a cycle C.

Delete the arcs of C from D to get a digraph $D - E(C)$ with fewer arcs: by the induction hypothesis, it has a cycle decomposition. Add C to that decomposition to obtain a cycle decomposition of D, completing the proof. □

Starting from a cycle decomposition, the algorithm for finding an Euler tour is the same as it was for undirected graphs. However, there is a further difficulty with stating the final result. We do not know exactly which graphs it applies to, because do not yet know what it means for a directed graph to be connected.

I will give you the statement first, and then we will determine what it means. This is the proper way to go about it: we figure out what we should prove by figuring out the assumptions we need to use.

Corollary 8.6. *A directed graph is Eulerian if and only if it is weakly connected (except possibly for isolated vertices) and all of its vertices are balanced.*

The proof is the same as the proof of Theorem 8.4: we start with a cycle decomposition, and splice the cycles into a partial Euler tour, one by one.

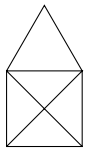
Question:	Looking at the proof of Theorem 8.4, where did we need the graph to be connected?
Answer:	We needed it to be impossible for the partial tour PT and the unused cycles $\mathcal{U}$ to have no vertices in common.

So the notion of connectedness we need is simply that it should be impossible to split the vertices of our digraph D into two sets, with no arcs between them in either direction: that's what we'd get if PT and the cycles of $\mathcal{U}$ shared no vertices. This is a notion of connectedness that ignores the orientations of the arcs.

Formally, we call a digraph D *weakly connected* if the underlying graph is connected. Now we know what Corollary 8.6 means!

8.6 Practice problems

1. The classic brainteaser in the "draw without lifting your pencil" genre is the picture of a stylized house shown below:

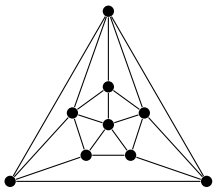


   In order to draw this picture without lifting your pencil or retracing any line, where must you start and end? (Of course, you can always swap the start and end.)

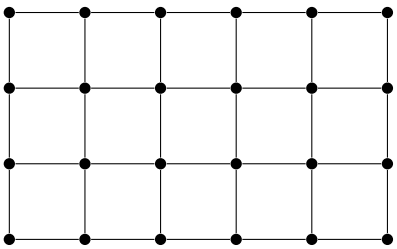
2. Find a cycle decomposition of the circulant graph $\mathrm{Ci}_{10}(1, 2)$, and use it to find an Euler tour of $\mathrm{Ci}_{10}(1, 2)$.

3. Find an Euler tour of the 4-dimensional hypercube graph, Q_4.

4. Let G be the graph below. Find a vertex x such that $G - x$ is Eulerian.



5. The graph below is not Eulerian, because some of its vertices are odd. What is the maximum length of a closed walk in this graph that does not use any edge more than once? Prove your answer.



6. In the complete graph K_7, every vertex has degree 6, so by Theorem 8.3, K_7 has a cycle decomposition.

   a) Suppose we wanted to decompose K_7 into as few cycles as possible. How many cycles would we want to use, and of what lengths? Give an example of such a decomposition.

   b) Suppose we wanted to decompose K_7 into as many cycles as possible. How many cycles would we want to use, and of what lengths? Give an example of such a decomposition.

7. Suppose that the complete graph K_n, where $95 < n < 105$, has a cycle decomposition in which every cycle has length 5. What must the value of n be?

8. Prove that every graph has a "double tour": a closed walk which uses every edge exactly two times, once in each direction. (That is, for every edge xy, the walk contains one segment $\ldots, x, y, \ldots$ and one segment $\ldots, y, x, \ldots$.)

9. This exercise presents part of a method due to Noga Alon [2] for finding perfect matchings (which we will cover in detail starting in Chapter 13) using Euler tours.

   a) Let G be a *bipartite graph*: that is, $V(G)$ is the union of two sets A and B, with $A \cap B = \varnothing$, such that all edges of G have one endpoint in A and one endpoint in B.

   Prove that if every vertex of G is even, then G has an even number of edges.

   b) Let G be a graph in which every vertex is even and (as in part (a)) the number of edges is even. Prove that G has a spanning subgraph H such that for every vertex x, $\deg_H(x) = \frac{1}{2}\deg_G(x)$.

   c) Prove by induction on k that for all $k \geq 0$, every 2^k-regular bipartite graph has a 1-regular spanning subgraph.

10. (BMO 1977) Prove that for each integer $n > 1$ it is possible to construct a necklace having $2n^2$ beads in all, these being of $2n$ different colors, in such a way that for each pair of different colors there is at least one pair of adjacent beads of these two colors. Is it possible to do the same using $2n^2 - 1$ beads in all? Give a reason for your answer.

Trees

9 Trees and spanning trees

The purpose of this chapter

Trees are some of the most fundamental objects in graph theory. There's no single big theorem that makes them useful; instead, there are many small facts. Trees appear when analyzing expression trees, or in computer science applications; they are useful in more advanced graph theory when analyzing connected graphs; they appear surprisingly often in recreational math. In this chapter, I want to motivate trees by introducing spanning trees, which is only one of many possible motivations.

It is natural to follow up by considering the problem of finding minimum-cost spanning trees, so I've included a section on that problem at the end of this chapter; though it is not crucial for any content in future chapters, it is nice to cover, if possible. (Students coming from a discrete math class might have already seen Prim's algorithm or Kruskal's algorithm; at the very least, the algorithm in this chapter is different, providing some variety.)

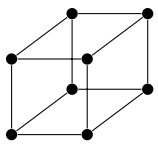
In this chapter and the next, I've also tried to include several diagrams of different trees, in particular the six in Figure 9.2. Take a few moments to just look at these and get a feeling for what trees are like, to supplement the mathematical properties that we will be proving.

9.1 Spanning trees

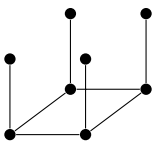
What does it take to connect a graph?

We have seen many examples of connected graphs. For example, the cube graph, shown in Figure 9.1a as a reminder, is connected.

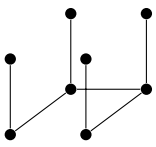
But not all the edges of the cube graph are necessary to have a connected graph. For example, we can remove all edges between the four vertices in the "top half" of the cube, and the result is still connected, because those vertices can still get to each other through the bottom half. The remaining graph, shown in Figure 9.1b, has only 8 edges.



(a) The cube graph



(b) From 12 edges to 8



(c) From 8 edges to 7

Figure 9.1: What does it take to connect the vertices of a cube graph?

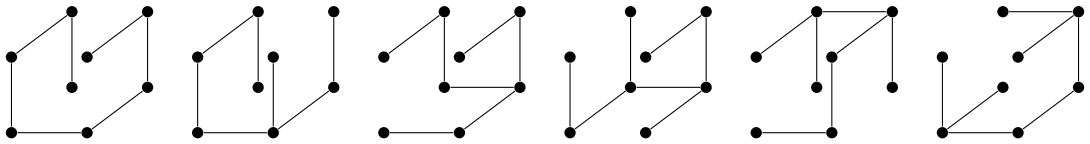


Figure 9.2: Spanning trees of the cube graph

Even that is not the best we can do. Remove any one of the edges in the bottom half, and the result is still connected: the bottom half of the cube is a path. This results in a 7-edge graph, shown in Figure 9.1c.

Finally, in this graph, we can check that none of the 7 edges remaining could be removed. This is certain to be true of whatever graph ends up our final stopping point; if it weren't, we would keep going. If we want to understand connected graphs, we would do well to start with graphs that have this property. They have a special name:

Definition 9.1. *A **tree** T is a minimally connected graph: T is connected, but for every edge $e \in E(T)$, the subgraph $T - e$ is no longer connected.*

(Why a "tree"? The term first appears in an 1857 paper of Arthur Cayley [13], whom we will meet again later on in our study of trees. Cayley's trees were diagrams used to represent various compositions of differential operators: diagrams which started from a root and branched out to smaller, similar diagrams, in just the way a tree grows. Graph-theoretically, they had the same structure as what we call a tree today.)

The graph in Figure 9.1c is a tree. Moreover, it is a spanning subgraph of the cube graph we started with in Figure 9.1a: we did not delete any vertices, only edges. In general, a spanning subgraph of a graph G which is a tree is a subgraph that connects all vertices of G using as few edges as possible; we call such an object a **spanning tree** of G.

It is natural to wonder whether some spanning trees are better or worse than others; after all, we did not arrive at Figure 9.1c with any attempt to make good decisions, but only did what is best in the moment. If you were to experiment different approaches to the cube graph, you would find that up to isomorphism, Figure 9.2 shows all possible spanning trees of the cube graph.

Question:	Which of these trees has the fewest edges?
Answer:	All six of them have 7 edges: in this case, anything we do must be equally good. In the next chapter, we will see that this is not a coincidence.

So why should we care about trees and spanning trees? For one, there are many practical applications, because spanning trees are exactly what you want if your goal is to connect a graph as cheaply as possible. Imagine, for example, a transportation company whose network spans an entire continent, but which has now fallen on hard times and needs to cut back. It would like to offer as few routes as possible, but it does not want to separate two parts of the country entirely. Under this circumstance, the transportation company's optimal solution will be a spanning tree of its old route network.

Spanning trees will also be useful for us theoretically, and the reason begins with the following theorem:

Theorem 9.1. *A graph G is connected if and only if it has a spanning tree.*

Proof. Let G be any connected graph. To find a spanning tree T of G, we will delete edges of G one at a time until we get a tree.

This can be done essentially any way we like. Suppose we have ended up at an intermediate graph H (some spanning subgraph of G) and H has an edge e such that $H - e$ is still connected. Then just delete edge e and keep going. (If there are many options for e, pick any of them.)

Eventually, we stop: we can't keep deleting edges forever, because G only has finitely many edges. However, the only way this process can stop is when deleting any edge would disconnect the remaining graph. That is exactly what it means to be a tree: we have arrived at a spanning tree of G.

In the other direction, suppose G has a spanning tree T. Then any two vertices x, y of G are also vertices of T, and T is connected, so there is an $x - y$ walk in T. That walk is also an $x - y$ walk in G, because T is a subgraph of G. Therefore G is connected. □

Question:	Does anything about this theorem change for multigraphs?
Answer:	No, and in fact, if we start with a multigraph, our very first step can be to delete loops and extra copies of edges to make it a simple graph. These deletions can never disconnect the graph.

The importance of Theorem 9.1 is that it tells us about a single object (a spanning tree) which is enough to show that a graph is connected. Without it, working directly from the definition, we would have to consider different pairs of vertices x, y in the graph, and find an $x - y$ walk between each of them.

This does not seem like a big deal at the moment, because to verify that a subgraph T is a spanning tree, we need to check that it is connected, which has all the same difficulties. (Finding $x - y$ walks in T might be even harder than it was in the original graph G.) Later on, as we learn more about trees, we will find ways to verify that T is a spanning tree which have nothing to do with being connected. Then, Theorem 9.1 will come into its full power.

A final use of spanning trees is for proving general results about connected graphs. If you have a problem about connected graphs that only gets easier to solve when the graphs have more edges, then you can begin by solving it in its hardest case: when the graph is a tree. If you do, Theorem 9.1 will immediately give you a general solution, by applying your specific solution to the spanning tree of a general connected graph.

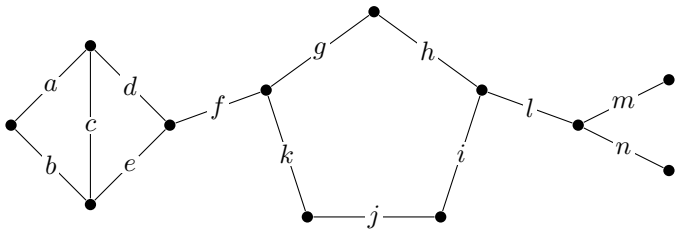


Figure 9.3: Which edges in this graph are bridges?

9.2 Bridges

Before we attain these promised powers, we need to learn much more about trees. Fortunately, there are lots of things to learn.

To begin with, let's take another look at the way we found a spanning tree in the proof of Theorem 9.1. We kept deleting edges if deleting them would not disconnect the graph, until there were no more edges left that we could delete. We can give a name to the type of edge that is left, as a prelude to studying such edges more thoroughly:

Definition 9.2. *A **bridge** e in a graph G is an edge of G such that $G - e$ has more connected components than G. Most commonly, G is a connected graph, in which case G is a bridge if and only if $G - e$ is not connected.*

As intuition for this name, you should imagine a long chain of islands like the Florida Keys, connected to the mainland only by a single bridge (the Overseas Highway, in the case of Florida). If anything happened to the bridge, then the islands would only be accessible by water.

Question:	If edge xy is a bridge of a connected graph G, how many connected components does $G - xy$ have?
Answer:	Two: each vertex z left in $G - xy$ either has a walk to y or to x. (If a $z - y$ walk in G no longer exists in $G - xy$, then that's because it relied on edge xy, which means that it must have at least reached x.)

Similarly, if G is not connected, and we delete an edge xy, then the number of connected components goes up by one: in G, there was a component H containing xy, which is replaced by the two components of $H - xy$.

Question:	In the graph shown in Figure 9.3, which edges are bridges?
Answer:	Edges f, l, m, and n.

Some graphs do not have any bridges at all. However, if we take a connected graph and start deleting edges from it, this might cause some of the remaining edges to become bridges. Eventually, if we keep deleting edges that are not bridges, we will only have bridges left. That's what it means to be a tree! An equivalent way to phrase the definition of a tree would be, "A tree is a connected graph in which all edges are bridges."

Question:	In Figure 9.3, what is the minimum effort needed to show that g is not a bridge?
Answer:	It's enough to show that it has a substitute: any walk in the graph that uses edge g could instead use the edges h, i, j, k. So it's impossible to find two vertices x and y such that all $x-y$ walks rely on the existence of edge g.

In other words, edges g, h, i, j, k make a cycle, and if something happens to one of these edges, you can always "go the long way" around the cycle. In fact, this is always the reason why an edge is not a bridge!

Lemma 9.2. *An edge xy of a graph G is a bridge if and only if it is not on any cycles.*

Proof. We may assume that G is connected. If not, pass to the connected component of G containing x and y; edge xy is still a bridge of that connected component.

For one direction of the proof, we must formalize the idea of "going the long way around". Take a cycle in G containing edge xy; we can represent it by a closed walk $(x_0, x_1, \ldots, x_{l-1}, x_0)$ where $x_0 = x$ and $x_{l-1} = y$. Then $(x_0, x_1, \ldots, x_{l-1})$ is an $x - y$ walk that does not use edge xy. We will use this walk to show that $G - xy$ is still connected, proving that xy is not a bridge.

Let s, t be two vertices of $G - xy$. Because G is connected, G has an $s - t$ walk. In it, if we replace every instance of $\ldots, x, y, \ldots$ by the $x - y$ walk we found, and every instance of $\ldots, y, x, \ldots$ by the reverse of this walk, we obtain an $s - t$ walk which does not use edge xy: it is still valid in $G - xy$. Since s and t were arbitrary, $G - xy$ is still connected.

For the other direction of the proof, suppose edge xy is not a bridge. Then $G - xy$ is still connected, and in particular, $G - xy$ contains an $x - y$ walk. By Theorem 3.1, G contains an $x - y$ path P. Adding edge xy to P produces a cycle (this is how the cycle graph C_n is defined from the path graph P_n), and that cycle contains edge xy. □

9.3 Properties of trees

What does Lemma 9.2 tell us about trees? In a tree T, every edge is a bridge, so no edge is part of any cycles. Therefore a tree T has no cycles at all.

Question:	When considering multigraphs, can a tree have loops or parallel edges?
Answer:	No: a loop is a cycle of length 1, and a pair of parallel edges is a cycle of length 2. So it is never necessary to model a tree as a multigraph rather than a simple graph.

Not all graphs without cycles are trees. However, the following is true:

Proposition 9.3. *A graph T is a tree if and only if it is connected and has no cycles.*

Proof. Both the condition in the proposition, and the definition of a tree, say that T is connected. So we must only show that a connected graph has no cycles if and only if all its edges are bridges (the second part of the definition of a tree).

If a graph has no cycles, then none of its edges lie on cycles: by Lemma 9.2, they are all bridges. Conversely, if all edges of a graph are bridges, then by Lemma 9.2, none of them lie on cycles—so no cycles can exist at all, because a cycle needs to contain some edges. This proves the equivalence we wanted. □

Proposition 9.3 is the first in a long line of results that characterize trees, giving an if-and-only-if condition for a graph to be a tree. Every such characterization could have been the definition of trees we started with, though some are more suited to it than others. Here is one more:

Proposition 9.4. *A graph T is a tree if and only if it has no cycles, but adding any edge would create a cycle.*

Proof. Suppose T is a tree; by Proposition 9.4, we already know it has no cycles. Let e be any edge we could add to T; we want to show that $T + e$ (the graph we get if we add edge e to T) has a cycle. Well, e cannot be a bridge of $T + e$: deleting it would only give us T again, and T is connected because it is a tree. So by Lemma 9.2, e must lie on some cycle in $T + e$, and in particular, adding e to T created a cycle.

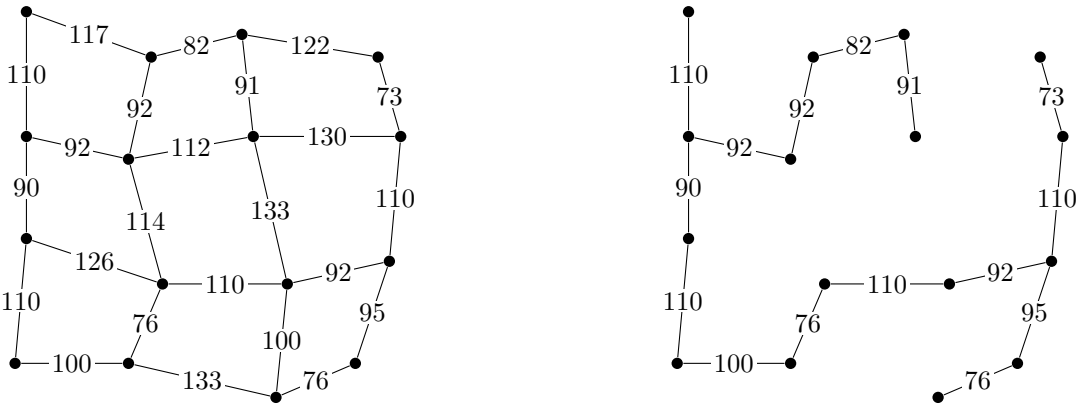
For the reverse direction, suppose that T has no cycles, but adding any edge would create a cycle. We want to prove that T is connected: then we can use Proposition 9.3 and conclude that T is a tree. To this end, let x, y be two vertices of T; we will be done if we find an $x - y$ path in T.

If xy is an edge of T, then there is such a path: a path of length 1. If not, then we know that $T + xy$ has a cycle. That cycle did not exist in T (because T has no cycles at all), so it must use the new edge xy. As in the proof of Lemma 9.2, when there is a cycle using edge xy, there is an $x - y$ walk not using edge xy that "goes the long way around" the cycle. This is an $x - y$ walk that still exists in T, finishing our proof that T is connected. □

We could rephrase Proposition 9.4 to say that trees are exactly the graphs which are "maximally acyclic": it has no cycles, but no more edges can be added without losing that property. In this way, it is a kind of opposite of our definition of trees as "minimally connected": connected with as few edges as possible, so that no more edges can be removed without losing that property.

9.4 Minimum-cost spanning trees

At the beginning of a chapter, I asked you to imagine a transportation company which wants to find the cheapest set of routes to keep its network connected. Though I said that the transportation company wants to find a spanning tree of its network, that's not the whole story. Not all spanning trees are equally cheap, because not all of the routes in the transportation network are equally cheap to run or to maintain. To keep track of this data, we need to consider more than just a graph.



(a) A weighted graph

(b) A minimum-cost spanning tree

Figure 9.4: Finding the minimum-cost spanning tree in a weighted graph

Definition 9.3. *A **weighted graph** is a graph G together with a function $c\colon E(G) \to [0, \infty)$. For an edge $e \in E(G)$, the value $c(e)$ is called the **cost** or the weight of e.*

Figure 9.4a shows an example of a weighted graph: what the transportation network's route map might look like. (To generate this simplified example, I placed the 16 vertices at slightly random points, and computed the distance between the points to determine the cost of the edges.)

When we measure the cost of a subgraph of a weighted graph, we compute the total cost of its edges, adding up their individual costs. A spanning tree T is a weighted graph is called a *minimum-cost spanning tree*, or MCST for short, if it has the minimum total cost. Figure 9.4b shows a minimum-cost spanning tree of the weighted graph in Figure 9.4a.

Question:	Should the MCST just use all the cheapest edges in the graph?
Answer:	No: some of the cheap edges might create a cycle with even cheaper edges, making them redundant. On the other hand, some very expensive edges might be necessary to connect the MCST.

There are many algorithms that can be used for finding the minimum-cost spanning tree of a weighted graph. Famous ones include Prim's algorithm and Kruskal's algorithm. In this chapter, I will show you the *reverse-delete algorithm* (published by Joseph Kruskal in the same paper [65] as the algorithm that bears his name), because it bears the most resemblance to our proof of Theorem 9.1.

In that proof, we repeatedly deleted edges, one at a time, until we were left with a spanning tree; however, the proof did not specify which edge to choose at each step. The only requirement is that we must never delete a bridge.

If we were to modify that strategy to find a minimum-cost spanning tree, which edges should we delete? The most natural guess is that of all the non-bridges, we should pick the most expensive: the one with the highest weight. This is a "greedy" strategy: it makes the best choice it sees in the moment, with no regard to how this affects future choices.

Question:	How could a greedy strategy fail—what should we be concerned about?
Answer:	It's conceivable that if we delete a cheaper edge and keep a more expensive one early on, then later in the process this will let us delete several more expensive edges to make up for it. The greedy strategy is not obviously correct: this will take some proof.

Let us summarize the reverse-delete algorithm formally. This is a slight change from the strategy we used to prove Theorem 9.1, because it only considers each edge once—however, there will be no point in returning to a previously-considered edge, because it will only be left alone if it is a bridge.

1. Let $e_1, e_2, \ldots, e_m$ be a list of the edges of G in descending order of cost: from most expensive to cheapest. Also, let $G_0 = G$; we will construct a sequence $G_1, G_2, \ldots, G_m$ of graphs as we go.
2. Starting at $i = 1$, look at edge e_i and ask: is e_i a bridge of G_{i-1}?

   If e_i is a bridge, set $G_i = G_{i-1}$; if e_i is not a bridge, set $G_i = G_{i-1} - e_i$.
3. Repeat step 2 for $i = 2, 3, \ldots, m$, until all edges have been considered. Return G_m as the minimum-cost spanning tree.

We will assume that there are no ties between the costs of the edges. (One of the problems at the end of this chapter will ask you to generalize this result to allow for ties.)

Theorem 9.5. *Let G be a connected weighted graph in which all edges have distinct costs. Then the output of the reverse-delete algorithm for G will be a minimum-cost spanning tree of G.*

Proof. The algorithm always produces a connected graph, because it starts with a connected graph and never deletes a bridge. Also, the only edges that are kept are bridges. Specifically, if edge e_i survives to the final graph G_m, it must first survive to G_i, which means it must have been a bridge of G_{i-1}. Therefore there is no cycle in G_{i-1} containing e_i. To obtain G_m from G_{i-1}, we only delete edges; therefore G_m also cannot have a cycle containing e_i, making e_i a bridge of G_m. Since this is true for every edge e_i, G_m must be a tree. Let's give G_m another name: call it T.

To prove that the algorithm is optimal, we need to compare T to an alternate spanning tree T', and somehow argue that T is better than T'. Specifically, what we'll do is prove that if $T' \neq T$, then T' cannot be the MCST, because we can make a small improvement to it. This will prove the theorem, because it leaves T as the only possible candidate for the MCST.

How can we do this? Well, first of all, if $T \neq T'$, we can find an edge e which is present in T, but not T'. (If every edge of T were present in T', then either T' would be equal to T, or T' would consist of T plus some additional edges. In the latter case, T' would not be a tree, because it would not be minimally connected.)

By Proposition 9.4, the graph $T' + e$ has a cycle. Let C be that cycle, and let e' be the most expensive edge of C.

By looking at the reverse-delete algorithm, we can prove that e' cannot be part of T, and in particular $e' \neq e$. Here's why: suppose that $e' = e_i$ in the list of edges by cost. In the graph G_{i-1} produced by the algorithm, e_i is still present, and so are all the other edges of C, because we have not gotten to any of them yet. Therefore C is a cycle of G_{i-1} containing e_i, meaning (by Lemma 9.2) that e_i is not a bridge of G_{i-1}. We conclude that e_i (that is, e') is deleted and does not survive to $T = G_m$.

Therefore we can modify T' as follows: add e, but then delete e'. This produces a cheaper connected graph!

Question:	Why is it cheaper?
Answer:	Because e' is the most expensive edge of C, and e is some other edge of C: we deleted a more expensive edge than we added.

Question:	Why is it still connected?
Answer:	Because we deleted an edge from a cycle of $T' + e$, which (by Lemma 9.2) was not a bridge.

Actually, it is true that $(T' + e) - e'$ is a tree itself, but proving that is inconvenient at the moment; it will be much easier once we can use Theorem 10.2 from the next chapter. However, we do not need to know that it is a tree. Since $(T' + e) - e'$ is connected, it certainly has a spanning tree by Theorem 9.1. The total cost of that spanning tree is at most the total cost of $(T' + e) - e'$, which in turn is strictly less than the cost of T'. Therefore T' cannot be the minimum-cost spanning tree.

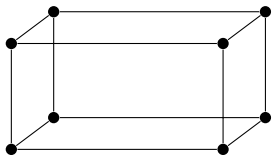
Since this is true of any spanning tree other than T, we conclude that T is the unique spanning tree of G. □

Question:	Will the tree T produced by the reverse-delete algorithm ever contain the most expensive edge, e_1?
Answer:	It might, if e_1 is a bridge of G. For example, if e_1 is the only edge incident to a vertex, then we are forced to use it no matter how expensive it is.

Question:	Will T always contain the cheapest edge, e_m?
Answer:	It will, though this is not as obvious. To delete e_m, it would need to be part of a cycle C in G_{m-1}. However, that cycle had other edges, which we considered before e_m. Those edges were also part of C when we considered them, so we would have deleted them, instead.

9.5 Practice problems

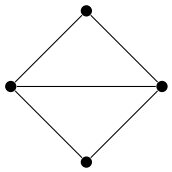
1. In the diagram of the cube graph shown below, each diagonal edge has length 1, each vertical edge has length 2, and each horizontal edge has length 4.



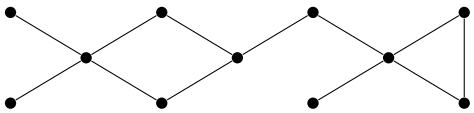
   Find a minimum-cost spanning tree of the cube graph, where the cost of each edge is taken to be its length in this diagram.

2. Generalizing the previous problem, suppose we make the hypercube graph Q_n into a weighted graph by giving an edge $\{b_1b_2\ldots b_n, c_1c_2\ldots c_n\}$ cost 2^k if $b_k \neq c_k$. (By definition of the hypercube graph, there will be only one position k where $b_1b_2\ldots b_n$ and $c_1c_2\ldots c_n$ disagree.) What will be the total cost of the minimum-cost spanning tree of this weighted graph?

3. Find all 8 spanning trees of the graph below.



4. Let G be the graph shown below:



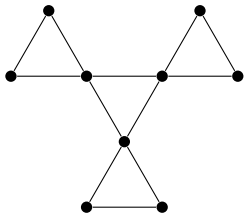
   a) Identify all the bridges in G. (There should be 5.)

   b) Find all the possible spanning trees of G.

5. Prove that an n-vertex graph with the degree sequence $n-1, 1, 1, \ldots, 1, 1$ must be a tree. What does such a graph look like?

6. Find a 3-regular graph G which has a bridge. (You'll need at least 10 vertices.)

7. Let G be the graph shown below:



   a) Find a spanning tree T of G whose *diameter* (the largest distance between any two vertices, as defined in Chapter 5) is as large as possible.

b) Is it possible to find two spanning trees of G that have only 3 edges in common? Give an example or explain why it is not possible.

8. a) Let G be a 4-regular graph in which edge xy is a bridge. Then $G - xy$ should have a connected component containing x, and a connected component containing y. Describe the degree sequences of each component.

   b) Conclude that a 4-regular graph cannot have a bridge.

9. Prove that T is a tree if and only if, between any two vertices of G, there is exactly one path. (This is another of the many characterizations of trees to accompany Proposition 9.3 and Proposition 9.4.)

10. Theorem 9.5 assumes that all edges of the graph have distinct costs: there are no ties.

    a) Prove that if there are ties between the costs of some edges, then we can still conclude one of two things in the proof: either we find a spanning tree cheaper than T' (so T' cannot be the MCST) or $(T' + e) - e'$ is a spanning tree with the same cost as T', but "closer" to T in some sense.

    b) Explain why this is enough to still deduce that T is an MCST (even if it might not be unique).

    c) Use this to prove that every spanning tree of the cube graph Q_3 must have 7 edges. (Do not, of course, use my claim that every spanning tree of Q_3 is isomorphic to one of the six trees in Figure 9.2.)

10 Properties of trees

The purpose of this chapter

Mathematically, the new properties of trees you will learn in this chapter are nothing special: in Theorem 10.1 and Theorem 10.2, we will prove how many edges they have, and later in Lemma 10.5 and Lemma 10.6, we will learn about their degree-1 vertices.

All this is notable because it is the point when trees suddenly begin pulling their weight as theoretical tools that help us solve problems. I have included two examples of this: the flower garden problem (based on my experience with recreational math) and Corollary 10.9 (based on my experience with competition math). In both cases, we are not solving a problem about trees: we are solving a problem that trees help us solve, because the problem is somehow related to a connected graph.

Lemma 10.6 is also important because it is what makes trees such an excellent setting for proofs by induction. (In the terminology of Appendix B, this lemma gives an inductive definition of trees.)

10.1 A square garden

Suppose you are designing a garden in the shape of an $n \times n$ grid. Each square of the grid can either contain flowers, or be part of a garden path for visitors. The garden path can bend and fork however you like, but it must be connected: visitors to the garden must be able to see all of it! Also, every square with flowers must be next to a square of the garden path—otherwise, visitors will not be able to see the flowers, and gardeners will not be able to water them. In both cases, squares that share only a corner don't count as being next to each other.

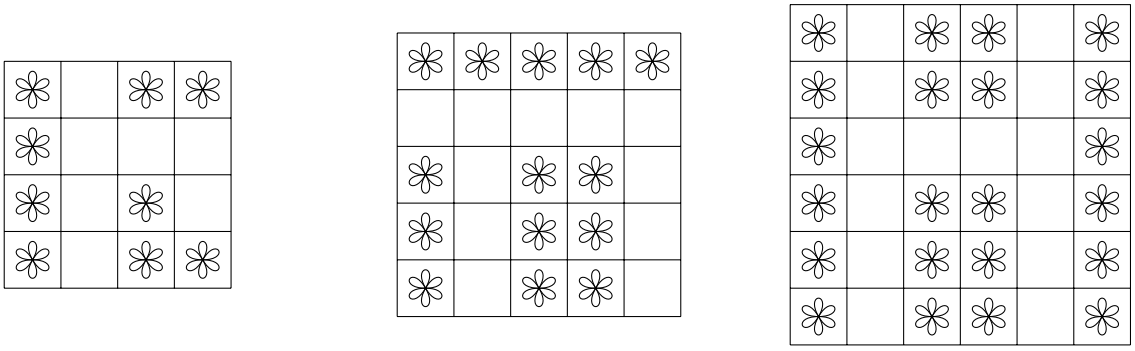
What is the largest number of squares of the grid that can contain flowers?

I have drawn some solutions for 4×4, 5×5, and 6×6 flower gardens in Figure 10.1, to show you what they can look like. If you'd like to try it yourself, see if you can manage to fill 29 squares of a 7×7 garden with flowers.

Solving the problem exactly for large $n \times n$ grids is, as far as I know, very difficult even with the help of a computer. However, with the aid of the material in this chapter, we will able to establish a very good upper bound on the number of flowers!

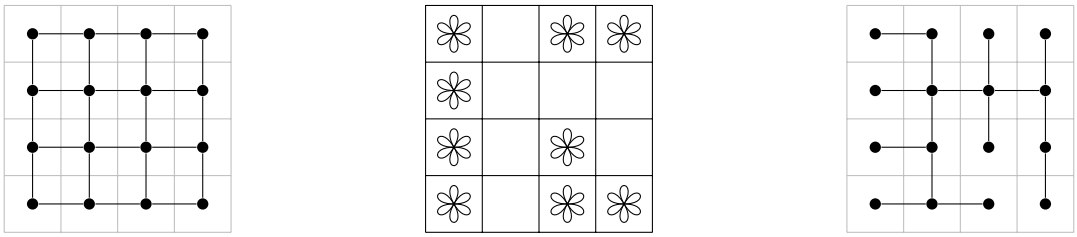
To do so, we will need a graph model. First, we must introduce the grid graph:

Definition 10.1. *For any $m \geq 1$ and $n \geq 1$, the **$m \times n$ grid graph $G(m, n)$** is the graph with mn vertices: points (x, y) where $x \in \{1, \ldots, m\}$ and $y \in \{1, \ldots, n\}$. Two vertices are adjacent in $G(m, n)$ when, viewed as points in the plane, they are distance 1 apart.*



(a) 4×4 (b) 5×5 (c) 6×6

Figure 10.1: Some high-density flower gardens



(a) 4×4 grid graph (b) 4×4 garden (c) The subgraph H

Figure 10.2: Representing a flower garden as a graph

The $n \times n$ grid graph $G(n, n)$ represents the $n \times n$ flower garden: its vertices correspond to the squares of the grid that forms the garden, and its edges correspond to adjacent squares. The $n = 4$ case of this is illustrated in Figure 10.2a. Of course, $G(n, n)$ does not tell us anything about the solution; it is merely the setting where the garden is posed. For simplicity, we will forget about the grid in favor of the grid graph: we will say that we plant flowers on some of the vertices of $G(n, n)$, and put a garden path on the rest.

We can think of a solution to the flower garden problem as a spanning subgraph H of $G(n, n)$. We include only the edges of $G(n, n)$ relevant to checking the solution. First, keep all edges of $G(n, n)$ between two vertices that are both part of of the garden path: these will be necessary to check that the garden path is connected. Second, for every vertex where a flower is planted, keep just one edge to an adjacent garden path vertex: these are necessary to check that the flower is accessible from the garden path. Figure 10.2b and Figure 10.2c show how a 4×4 solution turns into a subgraph H.

Every valid flower garden turns into a subgraph H, but which subgraphs are the best? They are the subgraphs with the most degree-1 vertices, because those are the vertices where flowers are planted. (In principle, a garden path vertex can also end up having degree 1, if it is an end of the garden path and not necessary to visit any flowers; however, in that case, we would do better to just plant a flower there, instead.)

In Chapter 4, we called a vertex with degree 1 a leaf. So we are looking for a connected spanning subgraph of H with as many leaf vertices as possible. Though there is no requirement for H to be a spanning tree, you may notice that the graph in Figure 10.2c is in fact a tree. This is no coincidence! Cycles in H can only appear in the garden path, but even then, they aren't

necessary, and we should avoid them: to plant more flowers, we want to have as little garden path as possible.

How do we maximize the number of leaves in the spanning tree? To do this, we need to combine some knowledge of vertex degrees with some facts about the number of edges in a tree, which we will prove later in this chapter. I will postpone discussing the graph theory behind the flower garden problem any further until you've seen that necessary background.

Before we move on to more theoretical questions, let me say a bit about the background of this problem. I made up the flower-garden formulation of it for this book, but I've seen many questions like it before. Mostly these have not been asked by professional mathematicians, but by people analyzing board games and video games with a square grid. However, while writing this chapter, I looked up and found a paper called "Maximum Leaf Spanning Tree Problem for Grid Graphs" by Li and Toulouse [69], studying this exact problem. The paper confirmed that my solutions for the 4×4 and 6×6 grid are optimal, and informed me that solving it is useful for practical questions in the areas of networking and circuit layout.

10.2 Counting edges in trees

By Proposition 9.3 from the previous chapter, a tree is a connected graph with no cycles. What can we say about the number of edges this requires?

In Chapter 4, we looked at the relationship between the number of edges in a graph, and cycles in the graph. Specifically, Corollary 4.7 tells us that an n-vertex graph with at least n edges is guaranteed to contain a cycle.

Question:	What does this tell us about the number of edges in an n-vertex tree?
Answer:	To avoid creating any cycles, the tree can have at most $n-1$ edges.

Question:	If an n-vertex graph has $n-1$ edges, must it be a tree?
Answer:	No: there's no reason to conclude either that the graph is connected or that it has no cycles. For example, we could start with a cycle that has $n-1$ vertices and edges, then add an isolated vertex: this is a graph with n vertices and $n-1$ edges which is not a tree.

If this were all we knew about trees, then we'd have to stop there, because n-vertex graphs without cycles can have any number of edges between 0 and $n-1$: there's nothing forcing them to have any edges. What forces trees to have edges, on the other hand, is the requirement to be connected: we need some edges to connect the tree! Let's explore how many.

Theorem 10.1. *A graph with n vertices and m edges has at least $n-m$ connected components.*

Proof. We'll prove this theorem by induction on m: just as in our proof of the handshake lemma (Lemma 4.1), we will see what happens as we add edges to a graph one at a time. This time, however, what we'll be paying attention to is the number of connected components.

Our base case is $m = 0$. If a graph has n vertices and 0 edges, then every vertex is an isolated vertex, so it is a connected component all by itself. There are always exactly $n = n - m$ components.

Now assume that the theorem is true for graphs with $m - 1$ edges, and let G be a graph with n vertices and m edges. Let xy be an arbitrary edge of G, and consider the $(m - 1)$-edge graph $G - xy$. By the induction hypothesis, $G - xy$ has at least $n - m + 1$ connected components.

When we add edge xy, this does not affect walks from any vertex not in the same connected component as x or y. Suppose that a vertex z has a walk in G that it did not have in $G - xy$: a walk of the form $(z, \dots, x, y, \dots)$ or $(z, \dots, y, x, \dots)$. Then if we end this walk as soon as it first reaches x or y, we get a $z - x$ or $z - y$ walk, so z must be in the same component as one of these vertices.

There are two cases for how this can go.

- If x and y are in the same connected component of $G - xy$, then adding edge xy does not do anything to the components at all. All the vertices affected are in the same component as x and y, and so there were already walks between them in $G - xy$; they have nothing to gain. Since $G - xy$ had at least $n - m + 1$ components, the same is true of G.
- If x and y are in different connected components of $G - xy$, then those two components become the same connected component of G. Any vertex in x's component can reach y (by going to x, then taking edge xy) and from there, it can reach any vertex in y's component.

  Since no other connected components are affected, G has one component less than $G - xy$: at least $n - m$ connected components.

In both cases, G has at least $n - m$ connected components, so the induction is complete and the statement we want is true for all n and m. □

We can use these ideas to prove a few more characterizations of trees. Let's do that, and summarize all our results so far in one big theorem:

Theorem 10.2. *The following conditions for a graph G with n vertices are all equivalent definitions of a tree:*

1. *G is minimally connected: it is connected, but deleting any edge will disconnect it.*
2. *G is maximally acyclic: it has no cycles, but adding any edge will create a cycle.*
3. *G is connected and has no cycles.*
4. *G is connected and has at most $n - 1$ edges.*
5. *G has no cycles and at least $n - 1$ edges.*
6. *G is uniquely connected: there is exactly one path between any two vertices.*

Proof. We already know that conditions 1, 2, and 3 are equivalent; we proved that in the previous chapter.

Let G be a graph satisfying these conditions. Then G has no cycles, so by the contrapositive of Corollary 4.7, G has at most $n-1$ edges. Also, G is connected, so by Theorem 10.1, it has at least $n-1$ edges. Now we know that condition 4 and condition 5 are also true.

This doesn't show that those conditions are equivalent to the first three. To prove that, we need the reverse implication as well: assuming only condition 4 or only condition 5, we need to prove one of the first three conditions.

Suppose that G satisfies condition 4. If we delete any edge from G, we will be left with at most $n-2$ edges, so by Theorem 10.1, we will be left with at least 2 connected components. Therefore G is connected, but deleting any edge will disconnect it: G satisfies condition 1.

Suppose instead that G satisfies condition 5. If we add any edge to G, we will get at least n edges, so by Corollary 4.7, we will get a cycle. Therefore G has no cycles, but adding any edge will create a cycle: G satisfies condition 2.

This shows that conditions 1–5 are equivalent. I've included condition 6 from a practice problem at the end of Chapter 9; it's also equivalent to the rest, but I won't prove that here. □

Question:	In condition 4 of Theorem 10.2, why do we say that G "has at most $n-1$ edges", when in fact we can conclude G has exactly $n-1$ edges?
Answer:	One of the reasons to have many characterizations of a tree is to make it as easy as possible to prove that a graph G is a tree. So conditions 4 and 5 are given weaker statements to make them easier to prove: if G is connected, and we can easily check that it cannot have more than $n-1$ edges, we don't need to check that it has exactly that many edges.

10.3 From trees to forests

The statement of Theorem 10.1 is an inequality: the number of connected components is at least $n-m$. In the case of a tree, $m = n-1$, and so the number of connected components is exactly $n-m$: it is 1. Are there other cases in which a graph with n vertices and m edges has exactly $n-m$ components?

If k is the number of connected components, then the inequality $k \geq n-m$ can be rephrased as $m \geq n-k$: a graph with n vertices and k connected components must have at least $n-k$ edges. To reach this lower bound, we want to try to use as few edges as possible: the bare minimum necessary in order to have the k components we want.

Question:	If we want to reach the minimum number of edges, what should we do in each connected component?
Answer:	We want each connected component to be a tree, because this uses the fewest edges to keep the component connected.

So we define:

Definition 10.2. *A **forest** is a graph in which every connected component is a tree.*

Question:	Using condition 3 of Theorem 10.2, we can rephrase this definition; how?
Answer:	This condition says that F is a forest if each connected component of F is connected and has no cycles. Well, of course each connected component is connected. Saying that each component has no cycles is the same as saying that F has no cycles at all! Therefore forests are the same as "acyclic graphs": graphs with no cycles.

Forests are also exactly the graphs for which the inequality of Theorem 10.1 becomes an equality. (This statements includes trees as a special case: in graph theory, a single tree is still a forest!)

Proposition 10.3. *A graph with n vertices and m edges has exactly $n-m$ connected components if and only if it is a forest.*

Proof. Suppose an n-vertex forest has k connected components: trees with $n_1, n_2, \ldots, n_k$ vertices, where $n_1 + n_2 + \cdots + n_k = n$. The i^{th} tree has $n_i - 1$ edges, so the total number of edges in the forest is

$$(n_1 - 1) + (n_2 - 1) + \cdots + (n_k - 1) = (n_1 + n_2 + \cdots + n_k) - k = n - k.$$

Since $m = n - k$, we have $k = n - m$, proving one direction of the proposition.

To prove the other direction, let G be a graph with n vertices, m edges, and exactly $n - m$ connected components, and suppose for the sake of contradiction that G is not a forest: some component of G is not a tree. Then G contains a cycle; let e be any edge on this cycle.

By Lemma 9.2, e is not a bridge, so $G - e$ has the same number of connected components as G: it has $n - m$ components. But $G - e$ has n vertices and $m - 1$, so by Theorem 10.1, $G - e$ must have at least $n - m + 1$ connected components: contradiction! So G must be a forest. □

10.4 Leaves in trees

To solve the flower garden problem, we were interested in the number of leaves (degree 1 vertices) of a spanning tree of the $n \times n$ grid graph. Now that we know the number of edges in a tree, the handshake lemma is sufficient to get a fairly precise upper bound.[15]

Proposition 10.4. *A connected subgraph of the $n \times n$ grid graph can have at most $\frac{2}{3}(n^2 + 1)$ leaves, and therefore an $n \times n$ flower garden can have at most $\frac{2}{3}(n^2 + 1)$ flowers.*

[15] I am going to state the result we get in a more general form, to allow for the possibility of cycles in the garden path and unused squares in the garden, but in spirit it is about spanning trees.

Proof. Let H be a connected subgraph of $G(n, n)$. We know that H has k vertices for some $k \leq n^2$, and by Theorem 10.1, we know that H has at least $k - 1$ edges.

Suppose that H has l leaves. The remaining $k - l$ vertices of H can have degree at most 4, because the vertices of $G(n, n)$ have degree at most 4. Therefore if we add up the degrees of all k vertices of H, we get a total of at most

$$\underbrace{1 + 1 + \cdots + 1}_{l \text{ times}} + \underbrace{4 + 4 + \cdots + 4}_{k-l \text{ times}} = l + 4(k - l) = 4k - 3l.$$

By the handshake lemma, this total must be equal to twice the number of edges in H, giving us the inequality

$$4k - 3l = 2|E(H)| \geq 2(k - 1).$$

Solving for l, we get $3l \leq 2k + 2$, or $l \leq \frac{2}{3}(k + 1)$. Since $k \leq n^2$, we obtain the bound we wanted: $l \leq \frac{2}{3}(n^2 + 1)$. □

The inequality in Proposition 10.4 is not quite the best possible: it is impossible to reach exactly $\frac{2}{3}(n^2+1)$ flowers. For example, the garden in Figure 10.1c has 22 flowers, while $\frac{2}{3}(6^2+1) = 24\frac{2}{3}$. (I have written the answer as a mixed integer, which is unusual in advanced math, to make it clear that the bound rounds down to 24.)

Part of the gap is due to number theory: actually, $\frac{2}{3}(n^2+1)$ is never an integer for any n, so the sharper bound $\frac{2}{3}n^2$ is also true. Part of the gap is that we can never obtain the ideal scenario where all vertices of H have degree 1 or 4. However, the true answer to the problem is always very close to $\frac{2}{3}n^2$, as I will ask you to prove in one of the practice problems at the end of this chapter.

In this application, we wanted to achieve the maximum number of leaves possible; it is also often useful to know what the minimum number of leaves is.

Lemma 10.5. *Every tree with at least two vertices has at least two leaves.*

Proof. Consider a tree with n vertices, l of which are leaves. When $n \geq 2$, it is impossible for the tree to contain degree-0 vertices: such a vertex is an isolated vertex, and cannot be part of a larger connected graph. Therefore the remaining $n - l$ vertices have degree at least 2.

We can now apply the handshake lemma, as we did in the proof of Proposition 10.4. We know that the sum of all n degrees in the tree is $2n - 2$: twice the number of edges. However, we also know that the sum is at least

$$\underbrace{1 + 1 + \cdots + 1}_{l \text{ times}} + \underbrace{2 + 2 + \cdots + 2}_{n-l \text{ times}} = l + 2(n - l) = 2n - l.$$

From the inequality $2n - l \leq 2n - 2$, we naturally deduce $l \geq 2$: the tree must have at least two leaves. □

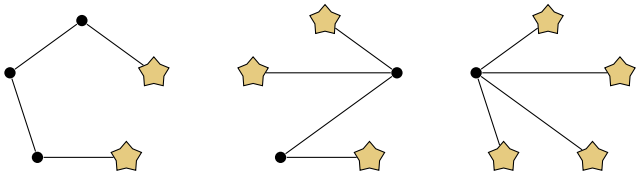


Figure 10.3: Leaves in the trees with 5 vertices

Question:	Why is it that we have a $\leq$ inequality ($2n - l \leq 2n - 2$) in this proof, but we had a $\geq$ inequality ($4k - 3l \geq 2k - 2$) in the proof of Proposition 10.4?
Answer:	In this proof, 2 is a lower bound on the degree of the non-leaf vertices; in the previous proof, 4 was an upper bound on the degree of the non-leaf vertices.

Figure 10.3 shows all three possible 5-vertex trees, up to isomorphism, with the leaves marked. You can see that the number of leaves can vary; in this case, it varies from 2 (on the left) to 4 (on the right).

Question:	Is it always possible to have an n-vertex tree with exactly 2 leaves?
Answer:	Yes: the path graph P_n is a tree, because it is connected and has $n-1$ edges, and only the start and end of the path are leaves.

Question:	For an n-vertex tree, what is the maximum number of leaves?
Answer:	It is $n-1$ (at least when $n > 2$): imitating the last tree in Figure 10.3, the graph with a single central vertex adjacent to $n-1$ leaves is an n-vertex tree. When $n > 2$, all n vertices cannot be leaves, because then the degree sum would be n, which is less than $2n-2$. However, this is achievable when $n = 2$: the only 2-vertex tree, P_2, has 2 leaves.

Since the path graph (the tree with as few leaves as possible) has a name, we should give the tree with as many leaves as possible a name, too.

Definition 10.3. *For all $n \geq 1$, the **star graph** S_n is the tree with $V(S_n) = \{1, 2, \ldots, n\}$ and $E(S_n) = \{12, 13, \ldots, 1n\}$.*

As with the path graph, not everyone can agree on whether the n-vertex star graph should be S_n (because it has n vertices) or S_{n-1} (because it has $n-1$ leaves), but I will go with the first option for the sake of consistency.

10.5 Induction on trees

Why is it important to have leaves? Because when a tree has leaves, we can pluck them to make the tree a tiny bit smaller.

Lemma 10.6. *If T is a tree and x is a leaf of T, then $T - x$ is also a tree.*

Proof. We have an abundance of conditions in Theorem 10.2 that we can check to prove this lemma. I think the easiest one to use is condition 5.

The graph $T - x$ has no cycles because it's a subgraph of T, which itself has no cycles. If T has n vertices and $n - 1$ edges, then $T - x$ has $n - 1$ vertices and $n - 2$ edges (because deleting x also deletes the single edge incident to x). The number of edges is one less than the number of vertices, so condition 5 of Theorem 10.2 is satisfied: $T - x$ is a tree. □

Lemma 10.6 is useful in a proof by induction. In Appendix B, I explain why it is important to write your induction proofs backward: having assumed the induction hypothesis for $(n-1)$-vertex graphs, we must consider an n-vertex graph and remove a vertex from it to apply the induction hypothesis. Lemma 10.6 gives us a convenient vertex to remove: if we are proving a theorem by induction about trees, then we can remove a leaf from an n-vertex tree to get an $(n-1)$-vertex tree.

Let me give you a couple examples of such proofs.

Our first example will eventually turn out to be silly: induction is not needed here. In Chapter 13, we will look at the property in this example again, and prove a more general result: Theorem 13.1. With this theorem, Proposition 10.7 will become a corollary with a one-line proof. For now, though, it is a good way to practice induction on trees.[16]

Proposition 10.7. *The vertices of every tree can be colored red and blue such that no two vertices of the same color are adjacent.*

Proof. We induct on n, the number of vertices in the tree. When $n = 1$, there is only one vertex, so we may give it any color we like without violating the condition.

Now assume, for some $n \geq 2$, that every $(n-1)$-vertex tree has a red-and-blue coloring in which no two vertices of the same color are adjacent. (Such a red-and-blue coloring is called a 2-*coloring*, as a special case of a definition from Chapter 19.) Let T be an arbitrary n-vertex tree; we will show that T also has a 2-coloring.

Let x be a leaf of T (which exists by Lemma 10.5), and let y be its only neighbor in T. By Lemma 10.6, $T - x$ is also a tree; it has $n - 1$ vertices, so by the induction hypothesis, it has a 2-coloring.

To color T, first give every vertex the same color that it had in $T - x$. Then, color x by the following rule: if y is red, color x blue, and if y is blue, color x red. This rule ensures that x and y do not have the same color; the same is true for every other pair of adjacent vertices, because they were already adjacent in $T - x$, and $T - x$ was given a 2-coloring.

[16] Later, the ability to prove this result much more easily will give you a well-deserved feeling of power.

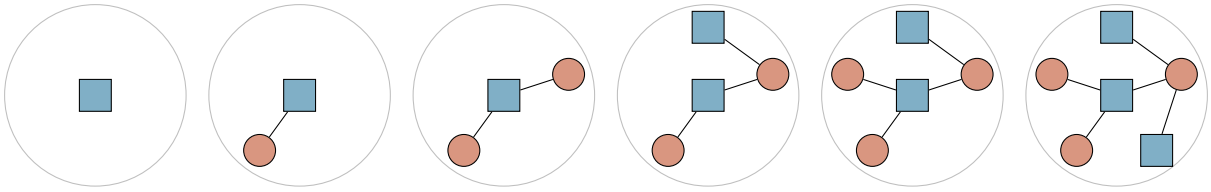


Figure 10.4: Coloring a tree using the proof of Proposition 10.7

This shows that T also has a 2-coloring, completing the induction step. By induction, trees with any number of vertices have 2-colorings, completing the proof. □

The proof of Proposition 10.7 is not just a proof: like many proofs by induction, it contains a recursive algorithm. Given an n-vertex tree, we can color it by choosing a leaf, removing it, and applying the coloring algorithm to the $(n-1)$-vertex tree that remains, before putting back the leaf we removed and coloring it as the n^{th} vertex. Although the recursive algorithm proceeds backwards from an n-vertex tree to a 1-vertex tree, if you think about the order in which vertices are colored, it will appear as though we started from 1 vertex and grew the tree by adding a succession of leaves, coloring them as we go. Figure 10.4 shows an example of coloring a 6-vertex tree using this strategy.

Question:	Can you use the proof to deduce how many 2-colorings a tree has?
Answer:	In the induction step of the proof, the color of x was forced: it had to be the opposite color of y. Similarly, at every previous step, the color of the leaf is forced. However, in the base case, we could give the single vertex of a 1-vertex tree either color. So there are two 2-colorings possible, based on which color we chose at that step. (They are opposites of each other: one is obtained from the other by switching red and blue.)

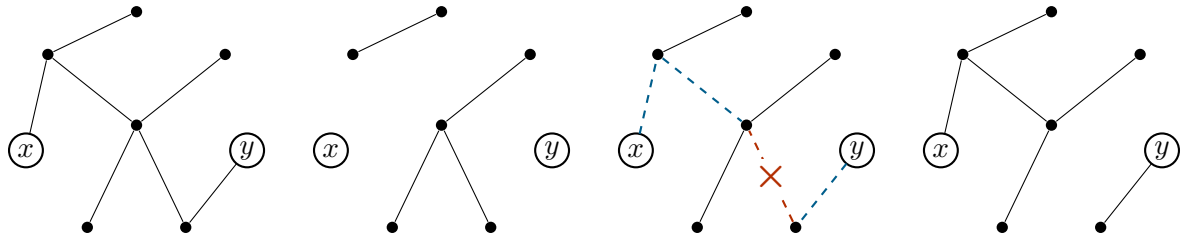
In most proofs by induction, the full power of Lemma 10.5 is not necessary: a single leaf is enough. Here is an example where we really must use the existence of two leaves.

Proposition 10.8. *Every tree with an even number of vertices has a spanning subgraph (not necessarily connected) in which every vertex has odd degree.*

Proof. As before, we induct on n, the number of vertices in the tree. Because the proposition only considers trees with an even number of vertices, our base case is $n = 2$. In a tree with 2 vertices, both vertices are leaves, so the tree itself is the spanning subgraph we need.

Now assume, for some even $n \geq 4$, that the proposition is true for all $(n-2)$-vertex trees. (We go back from n to $n-2$, the previous even number.) Let T be an arbitrary n-vertex tree; by Lemma 10.5, T has two leaves, which we call x and y. Figure 10.5a shows an example of such a tree, in which you can follow along as we carry out the induction step.

Applying Lemma 10.6 twice, the subtree $(T-x)-y$ is an $(n-2)$-vertex tree. So it has a spanning subgraph in which every vertex has odd degree. Add x and y back into this subgraph (as isolated vertices) and call the result H. (The graph H in our example is shown in Figure 10.5b.)



(a) T; leaves x and y (b) The subgraph H (c) Toggling along P (d) The final result, H'

Figure 10.5: The induction step of Proposition 10.8

This H is a spanning subgraph of T, but what about the degrees? Vertices x and y have degree 0 in H, by construction, and 0 is an even number. However, every other vertex has an odd degree in H, just as we wanted. All that's necessary is to fix x and y.

At first, this seems impossible. Suppose you change H, adding the unique edge incident to x. Now vertex x has odd degree, but its neighbor has even degree. If you make a change to fix that neighbor, another vertex will go from odd to even, and so on.

This is why we need to work with two leaves at once. In T, there is an $x - y$ path P. Change H to a new graph H' by toggling every edge along P. That is, for every edge of P, if it is not in H, add it, and if it is in H, remove it. The toggled edges are shown in Figure 10.5c, with the final result H' shown in Figure 10.5d.

(In Chapter 14, we will define this as the *symmetric difference* operation: $H' = H \oplus P$. In the meantime, if the explanation is not clear, rely on Figure 10.5)

We can check by cases that every degree in H' is odd:

- A vertex not in $V(P)$ is untouched, and still has odd degree.
- Vertices x and y gain an edge (because the edges incident to x and to y were not in H before), so their degree goes from 0 to 1: an odd number.
- Finally, a vertex in $V(P)$ other than x and y either gained two edges, gained an edge and lost an edge, or lost two edges. The change in degree is $+2$, $+0$, or -2, so the degree remains odd.

Therefore T has a spanning subgraph in which every vertex has odd degree: the subgraph H'. By induction, the same is true for every tree with an even number of vertices. □

Question:	Can a tree with an odd number of vertices have a spanning subgraph such as the one in Proposition 10.8?
Answer:	No: that would be a graph with an odd number of vertices, all with odd degree, which cannot exist by Corollary 4.2 to the handshake lemma.

This example is also an illustration of an idea that I first mentioned in Chapter 9: if we must prove a theorem for connected graphs, and the theorem is only helped by having more edges, then it's enough to prove the theorem for trees—which will be the hardest case.

Here, once we have Proposition 10.8, we may immediately obtain the following corollary, which was proven by Atsushi Amahashi in 1985 [3], and later appeared as a problem on the 2005 Bay Area Mathematical Olympiad [7].

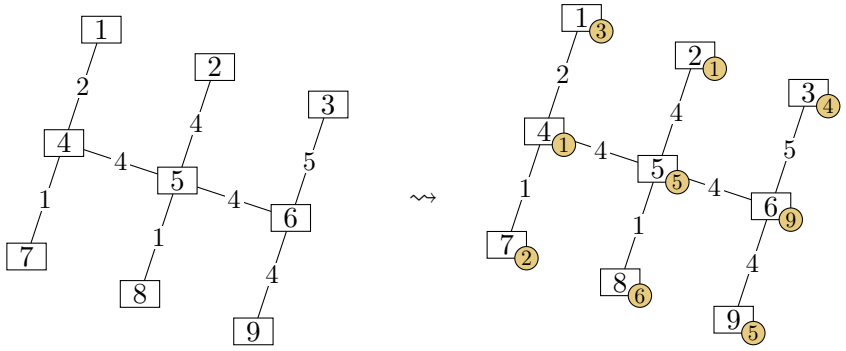
Corollary 10.9. *Every connected graph with an even number of vertices has a spanning subgraph (not necessarily connected) in which every vertex has odd degree.*

Proof. Apply Proposition 10.8 to a spanning tree of the graph. □

10.6 Practice problems

1. Let T be tree whose degree sequence has the form $4, 3, 2, 1, 1, 1, \ldots$ (that is, $4, 3, 2$ followed by some number of 1's).
   a) Determine the number of 1's in the degree sequence of T.
   b) There is more than one possibility for a tree T with this degree sequence. Give two non-isomorphic trees with this degree sequence, and explain why they are not isomorphic.
2. Find all 6-vertex trees up to isomorphism. (There are six of them.)
3. Let G be a graph with 10 vertices and 10 edges.
   a) If G contains exactly one cycle, how many connected components must it have? Give an example of such a graph.
   b) If G contains exactly two cycles, how many connected components must it have? Give an example of such a graph.
   c) If G contains exactly three cycles, there's two possible values for the number of connected components. Why is that? Give examples for each possibility.
4. Prove that when n is a multiple of 3, there is a solution to the $n \times n$ flower garden problem which contains at least $\frac{2}{3}(n^2 - n) + 2$ flowers.

   (You will have to come up with a generalizable layout for the flower garden which contains this many flowers. See if you can generalize the layout in Figure 10.1c.)
5. a) Prove that if x and y are vertices of a graph G in the same connected component, then adding edge xy to G creates at least one new cycle.
   b) Imitate the proof of Theorem 10.1 to prove that a graph with n vertices and m edges has at least $m - n + 1$ cycles.
6. a) Let T be a weighted tree in which the cost $c(e)$ of every edge is a positive integer. Prove that it's possible to choose a positive integer value $f(x)$ for each vertex x such that for all edges $xy \in E(T)$, $c(xy) = |f(x) - f(y)|$. An example of being given such a problem and solving it is shown below:



b) Find an example of a weighted graph (not a tree!) where every cost $c(e)$ is a positive integer, but the task in part (a) is not possible.

7. Prove that if T is a k-vertex tree and G is a graph with minimum degree at least $k-1$, then G contains a copy of T.

8. In Chapter 6, we had a rather complicated procedure for determining whether there is a graph with a given degree sequence. If we want to know whether a tree with a given degree sequence exists, the problem is much easier!

   Prove that for any sequence of $n \geq 2$ positive integers whose sum is $2(n-1)$, there is a tree with that degree sequence.

9. Prove that a 99-vertex tree cannot have two vertices of degree 50.

10. Let T_1 and T_2 be two trees such that $T_1 \cap T_2$ (the graph whose vertices and edges are exactly the vertices and edges present in both T_1 and T_2) is also a tree. Prove that the union $T_1 \cup T_2$ is a tree.

11. (APMO 2010) Let n be a positive integer. n people take part in a certain party. For any pair of the participants, either the two are acquainted with each other or they are not. What is the maximum possible number of the pairs for which the two are not acquainted but have a common acquaintance among the participants?

11 Cayley's formula

The purpose of this chapter

I am including a chapter on Cayley's formula for the number of labeled n-vertex trees in this book, first of all, because it is a beautiful piece of mathematics. It comes with an introductory look at counting problems in graph theory, which I think is especially important to include because I often tend to overlook such problems when I think about what a graph theorist needs to know. (The next chapter is also about counting problems in graph theory, but from a very different perspective.) In short, even though Cayley's formula is not necessary as a prerequisite for any of the following chapters, I think it is a worthwhile topic to study on its own.

I do not like including sections such as the last section of this chapter, which is mainly about results too complicated to prove in this book. However, I think that a textbook must say something about counting trees up to isomorphism, even if it simply says that not very much is known about the problem. When teaching this topic, I would not have feelings that are nearly as strong; I think it is perfectly acceptable to say that the textbook has a page on this version of the problem, and leave it at that.

11.1 How to count graphs

Graph enumeration is a whole discipline of graph theory, but we have a lot of ground to cover in this textbook. As a result, this is the first and only chapter in which we will seriously look at problems about counting graphs satisfying a given property.

When we count graphs, we must first choose between two options that are typically described as "counting labeled graphs" and "counting unlabeled graphs". This is a bit misleading, because all graphs are labeled, in the sense that all graphs have a vertex set with distinguishable elements. Here is how to describe these options more precisely.

Counting labeled graphs is typically the easier of the two problems. To describe it more precisely, suppose that we being by choosing a set V of size n: it does not matter too much which set V, as long as we choose it, so choosing $V = \{1, 2, \ldots, n\}$ is a very reasonable and simple choice. Then we ask: how many graphs with vertex set V have a certain property?

The other option is to count graphs up to isomorphism, and this is the problem typically called "counting unlabeled graphs". Here, we start with the set $\mathcal{S}$ of all labeled graphs of that type (again, with some fixed vertex set V). In most problems, some of the graphs in $\mathcal{S}$ will be isomorphic. Graph isomorphism is an equivalence relation on $\mathcal{S}$, so it can be used to split $\mathcal{S}$ into equivalence classes. When we count graphs up to isomorphism, we ask: how many equivalence classes are there?

In this chapter, we will count n-vertex trees. Most of the chapter is devoted to the labeled counting problem, for which a clean and beautiful formula exists. Much less is known about the unlabeled counting problem, but I will summarize what is known about it at the end of the chapter.

For either counting problem, one of the most important tools graph theorists use is counting by bijection. If a function $f\colon X \to Y$ is a bijection, then the sets X and Y have the same size: to a set theorist, this is actually how the size of a set is defined. Our bijections will be bijections from a set of graphs to a set of sequences, and so we will think of them somewhat differently: we will call them *encoding schemes*.

An encoding scheme is a rule by which we can take a graph (or whatever else) and write down a sequence of numbers to represent it. This is a very natural idea in the modern day, when every piece of information has an encoding scheme: after all, a computer must represent all the data it works with as by sequences of zeroes and ones. That's exactly the way you should think about encoding schemes.

We will want to know two things about our encoding schemes. These correspond to the definition of a bijection, though we won't invoke that definition directly.

- We will want to make sure that the encoding scheme is a *unique encoding scheme*: from the sequence it produces as an output, we can uniquely determine which graph was the input. It's nice if we have a simple algorithm to find the graph, but all that's strictly necessary is to know that two different graphs never produce the same sequence.
- We will want to know which sequences of numbers are *valid encodings*: which ones

An encoding scheme can be described as a function f from graphs to encodings, and both of the properties above can be phrased in terms of f.

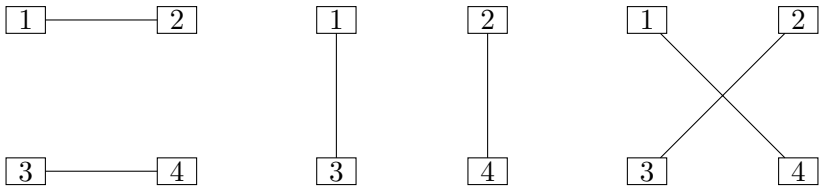
Question:	If the encoding scheme is unique, what does that say about f?
Answer:	It says that f is an injective (one-to-one) function.

Question:	If every element of some set S is a valid encoding, what does that say about f?
Answer:	It says that f is a surjective (onto) function, at least if it is a function to the set S.

So that's the connection to bijections. For our purposes, the consequence is this: once we verify that we have a unique encoding scheme of a set of graphs, we can count the graphs in that set by counting the valid encodings.

To avoid getting too abstract about it, here are two examples.

Problem 11.1. *How many graphs with vertex set* $\{1, 2, \ldots, n\}$ *are there?*



(a) $12, 34$ (b) $13, 24$ (c) $14, 23$

Figure 11.1: The 1-regular graphs with vertex set $\{1, 2, 3, 4\}$

Answer to Problem 11.1. We will encode these graphs as binary strings of length $\binom{n}{2}$. Here, $\binom{n}{2} = \frac{n(n-1)}{2}$ is the number of edges in the complete graph K_n (which we also consider to have vertex set $\{1, 2, \ldots, n\}$). To encode a graph G with $V(G) = \{1, 2, \ldots, n\}$, we go through the edges of K_n in a fixed order, such as the dictionary order

$$12, 13, \ldots, 1n, 23, 24, \ldots, 2n, \ldots.$$

For each edge, we ask: does G also contain that edge? If so, we write down a 1; if not, we write down a 0.

From the output of this procedure, we can look through the bits one at a time and uniquely determine the edge set $E(G)$, so this is a unique encoding scheme. Starting from an arbitrary sequence of $\binom{n}{2}$ zeroes and ones, we can reconstruct a graph that will give that sequence, so every one of these sequences is a valid encoding.

There are $2^{\binom{n}{2}}$ sequences of $\binom{n}{2}$ zeroes and ones (because we have 2 choices for each bit in the sequence) and therefore there are $2^{\binom{n}{2}}$ graphs with vertex set $\{1, 2, \ldots, n\}$. □

Problem 11.2. *How many* 1*-regular graphs with vertex set* $\{1, 2, \ldots, n\}$ *are there?*

Answer to Problem 11.2. As we know from Chapter 5, such graphs only exist when n is even. In the cases of even n, we could begin by listing all $n/2$ edges in the graph. This is done in Figure 11.1 in the case $n = 4$.

Question:	Are there multiple ways to list the edges in a graph in a sequence?
Answer:	Yes: for each edge xy, we can also write it as yx, and we can also write the edges in any order.

To make the encoding scheme unique, we should make a rule for how the edges should be written, and in which order they should appear. A simple option is to write a "sorted sequence of sorted edges": to sort each edge xy so that $x < y$, and to sort the sequence of edges by the first number in each pair. This is already done in Figure 11.1.

Question:	Are all "sorted sequences of $n/2$ sorted edges" valid encodings?
Answer:	No: some of them encode graphs with $n/2$ edges that are not 1-regular, by leaving out some vertices and using others too many times. Each vertex must appear exactly once.

If not all sequences are valid encodings, this complicates our life, but we can still proceed. We just have to describe which encodings are valid, and count only the valid ones.

Question:	Going from left to right, what are the constraints on the vertices in the "sorted sequence of sorted edges"?
Answer:	The first vertex of each sorted edge is uniquely determined: it is simply the smallest element of $\{1, 2, \ldots, n\}$ that has not yet appeared. The second vertex of each sorted edge can be any of the vertices that have not yet appeared.

Question:	How many possibilities are there for each vertices, given the vertices to its left?
Answer:	There is only 1 option for the first vertex of each sorted edge. For the second vertex in the k^{th} sorted edge, there are $n - 2k + 1$ options, because $2k - 1$ vertices have already appeared.

In such a case, we can count by the product principle, multiplying the numbers of possible vertices in each position. We can do so because that number of options is always the same for a given position, no matter which vertices came before. (Always make sure that this is true before applying the product principle!)

Multiplying together the numbers of possible vertices in each position, we get the product

$$\underbrace{1 \cdot (2n-1) \cdot 1 \cdot (2n-3) \cdot 1 \cdot (2n-5) \cdots 1 \cdot 3 \cdot 1 \cdot 1}_{n \text{ factors}}$$

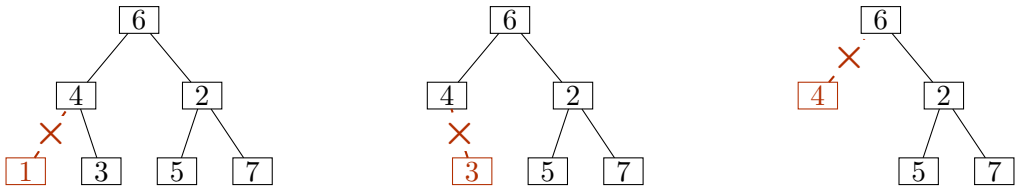
in answer to the problem. We can ignore the factors of 1 and just say that this is the product of the first $n/2$ odd positive integers. For example, we get $3 \cdot 1 = 3$ when $n = 4$ (as seen in Figure 11.1), $5 \cdot 3 \cdot 1 = 15$ when $n = 6$, $7 \cdot 5 \cdot 3 \cdot 1 = 105$ when $n = 8$, and so on. This product is often written with the double factorial symbol $(2n-1)!!$. □

11.2 Trees and deletion sequences

Having solved a few problems for practice, we can move on to the question we're really interested in: how many trees with vertex set $\{1, 2, \ldots, n\}$ are there?

The solution to Problem 11.2 gives us a good starting point. An n-vertex tree, as we know, is in particular an $(n-1)$-vertex graph. So we can begin by writing down the edges of that graph, in a convenient order.

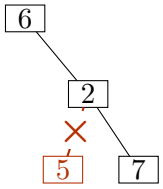
The most convenient order to use here is not the dictionary order we used before. We would like to make use of the structure of a tree! In particular, we know from Lemma 10.6 in the previous chapter that if we remove a leaf vertex (and its only edge) from a tree, we get a smaller tree. We used this to great effect to write inductive proofs of theorems about trees; it can be used to equally great effect to write recursive algorithms for problems about trees.



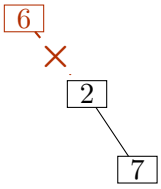
(a) Write down $(1, 4)$

(b) Write down $(3, 4)$

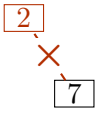
(c) Write down $(4, 6)$



(d) Write down $(5, 2)$



(e) Write down $(6, 2)$



(f) Write down $(2, 7)$

Figure 11.2: Finding the deletion sequence of a tree

Given a tree T with vertex set $\{1, 2, \ldots, n\}$, here is an algorithm to list all its edges in a uniquely defined order:

1. If $n = 1$, then there are no edges, so the list of edges should be empty as well.
2. Otherwise, for $n > 1$, the tree T will have some leaves. Let x be the lowest numbered leaf, and let y be its neighbor: write down the pair (x, y), in that order.
3. Delete vertex x and edge xy from T to get an $(n-1)$-vertex tree $T - x$. To write the rest of the sequence, go back to step 1, but with tree $T - x$ in place of T.

We will call the result of this algorithm a *deletion sequence*[17] for T. As an illustration of the technique, Figure 11.2 shows how, starting with the tree in Figure 11.2a, we determine that its deletion sequence is

$$(1, 4),\ (3, 4),\ (4, 6),\ (5, 2),\ (6, 2),\ (2, 7).$$

Question:	Is this a unique encoding scheme?
Answer:	Yes, it is. We can recover the tree from its deletion sequence, because the deletion sequence is after all a list of edges. Moreover, each tree only has one deletion sequence, because the deletion sequence is computed by an algorithm with no freedom at any step.

There is a great deal of redundancy in the deletion sequence of a tree. Before proving a general result about it, let's explore a few examples. In all of these, I will erase some numbers from the deletion sequence we just constructed and ask how they can be filled back in to get a valid deletion sequence. Of course, one way to fill in the number is to put back the number we erased, getting back the deletion sequence we started with. However, we want to know if there are any other deletion sequences that have a different value in that blank!

[17] This is not a universally recognized term, but simply the term I will use in this chapter to explain our counting strategy.

Question: In the incomplete deletion sequence

$$(1,4),\ (3,4),\ (4,6),\ (5,2),\ (6,2),\ (2,_),$$

how many ways are there to fill in the blank?

Answer: The number in the blank can only be 7.

The reason is that vertex 7 will never be deleted as the smallest leaf of the tree: there are always at least two leaves, one of which is smaller than 7. Therefore it is the last vertex remaining, and will always occupy the last position in the deletion sequence.

Question: In the incomplete deletion sequence

$$(1,4),\ (3,4),\ (_,6),\ (5,2),\ (6,2),\ (2,7),$$

how many ways are there to fill in the blank?

Answer: The number in the blank can only be 4.

The reason is that vertices $1, 2, 3, 4, 5, 6$ must all be eventually deleted. The five complete ordered pairs tell us when we deleted vertices 1, 2, 3, 5, and 6, so the incomplete ordered pair must tell us when we deleted vertex 4.

Question: In the incomplete deletion sequence

$$(1,4),\ (3,4),\ (_,6),\ (_,2),\ (6,2),\ (2,7),$$

how many ways are there to fill in the two blanks?

Answer: Once again, the answer is uniquely determined: the blanks must contain 4 and 5, in that order.

By the same reasoning as above, we know that 4 and 5 must go in those two blanks in some order. Since neither number appears later in the deletion sequence, we know that none of their neighbors are deleted later: after the first two steps, both vertices 4 and 5 are leaves. Since 4 is a smaller number, it will be the leaf deleted first.

But all of these questions are thinking too small: we can take the incomplete sequence

$$(_,4),\ (_,4),\ (_,6),\ (_,2),\ (_,2),\ (_,_)$$

and fill in all 7 blanks in a unique way!

First of all, we know that the first six blanks are the numbers 1 through 6, while the last blank is 7. The order of the numbers 1 through 6 is not known yet, but even without knowing the order, we know how many times each number appears in the deletion sequence, in total: the number of times we see it in the incomplete sequence, plus 1.

This tells us the degree of every vertex:

x	1	2	3	4	5	6	7
$\deg(x)$	1	3	1	3	1	2	1

Why is this helpful? Because it tells us that at the beginning of the algorithm to generate the deletion sequence, the leaves were 1, 3, 5, and 7. Of these, 1 is the smallest, so it must be the first leaf deleted: the first pair is $(1, 4)$.

From there, we can deduce that after vertex 1 is deleted, the degrees of the vertices were as follows:

x	2	3	4	5	6	7
$\deg(x)$	3	1	2	1	2	1

The leaves were 3, 5, and 7, of which 3 is the smallest. Therefore the second pair must be $(3, 4)$.

If we keep going in this manner, we can reconstruct the entire deletion sequence, because we can determine the degree of each remaining vertex at each step of the algorithm. Only 5 of the 12 numbers in the deletion sequence were necessary!

11.3 Prüfer codes

Our strategy in the preceding section generalizes fully. We can summarize the properties we use in the following two properties of a deletion sequence $(a_1, b_1), (a_2, b_2), \ldots, (a_{n-1}, b_{n-1})$:

$$\text{For every } k \text{ from 1 to } n-1, \text{ the number } a_k \text{ is the smallest positive number not contained in the set } \{a_1, a_2, \ldots, a_{k-1}\} \cup \{b_k, b_{k+1}, \ldots, b_{n-2}\}. \tag{11.1}$$

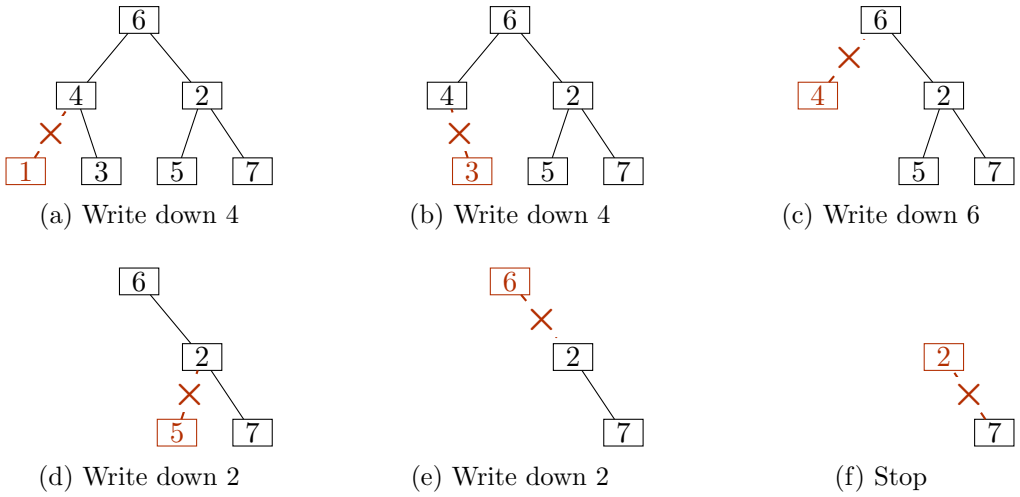
$$b_{n-1} = n. \tag{11.2}$$

Lemma 11.1. *If a sequence $(a_1, b_1), (a_2, b_2), \ldots, (a_{n-1}, b_{n-1})$ is the deletion sequence of a tree with vertices $1, 2, \ldots, n$, then it satisfies properties (11.1) and (11.2).*

Proof. The tree at every stage of the algorithm that generates the deletion sequence has at least two leaves, so the smallest leaf will never be vertex n. Therefore vertex n will never be the deleted leaf, so it will be the last vertex remaining: $b_{n-1} = n$. This proves (11.2).

Meanwhile, $a_1, \ldots, a_{n-1}$ are a permutation of $1, 2, \ldots, n-1$, representing the order in which the other vertices are deleted. After the first $k-1$ stages of the algorithm that generates the deletion sequence, the tree that remains has edges (a_k, b_k) through (a_{n-1}, b_{n-1}). Vertices in the set $\{a_1, a_2, \ldots, a_{k-1}\}$ are not present in this tree; they have already been deleted.

The remaining vertices of the tree appear once in the set $\{a_k, a_{k+1}, \ldots, a_{n-1}, b_{n-1}\}$. They have degree 1 if this is their only appearance: if they do not appear in the set $\{b_k, b_{k+1}, \ldots, b_{n-2}\}$. Vertex a_k is the smallest leaf remaining at this stage, so it is the smallest positive number not contained in the set $\{a_1, a_2, \ldots, a_{k-1}\} \cup \{b_k, b_{k+1}, \ldots, b_{n-2}\}$. This proves (11.1). □



(a) Write down 4 (b) Write down 4 (c) Write down 6

(d) Write down 2 (e) Write down 2 (f) Stop

Figure 11.3: Finding the Prüfer code of a tree

Since the numbers determined by Lemma 11.1 can be deduced from the others, they are not necessary to recover the tree. Therefore, instead of recording the deletion sequence of a tree, it is enough to record the sequence $(b_1, b_2, \ldots, b_{n-2})$. This sequence is called the **Prüfer code** of the tree, named after Heinz Prüfer, who proposed Prüfer codes as a method of counting labeled trees in 1918 [87].

It is worth mentioning that the Prüfer code of a tree can be directly computed using an abbreviated version of the algorithm that computed the deletion sequence. The only two differences are that instead of writing down the pair (x, y), we only write down the vertex y, and that we stop when there are 2 vertices and 1 edge left. Figure 11.3 illustrates this on an example.

Question:	Are Prüfer codes a unique encoding scheme?
Answer:	Yes: from the Prüfer code, we can recover the deletion sequence using properties (11.1) and (11.2), and the deletion sequence simply tells us the edges of the tree.

Question:	Is this enough to count the with vertex set $\{1, 2, \ldots, n\}$?
Answer:	No: we need to know whether all possible sequences $(b_1, b_2, \ldots, b_{n-2})$ are valid Prüfer codes, or whether there are some constraints on these numbers.

In fact, there are no further constraints: every sequence $(b_1, b_2, \ldots, b_{n-2})$, where each term is an element of $\{1, 2, \ldots, n\}$, is the Prüfer code of a tree with vertex set $\{1, 2, \ldots, n\}$. To prove this, the most important claim we have not yet shown is that (11.1) and (11.2) are not just properties every deletion sequence has: they are properties only a deletion sequence can have.

Lemma 11.2. *If a sequence $(a_1, b_1), (a_2, b_2), \ldots, (a_{n-1}, b_{n-1})$ in which every term (a_i, b_i) is a pair of numbers from 1 to n satisfies properties (11.1) and (11.2), then it is the deletion sequence of a tree with vertex set $\{1, 2, \ldots, n\}$.*

Proof. Let's make some initial observations. First, since $a_k \notin \{a_1, a_2, \ldots, a_{k-1}\}$, the numbers $a_1, a_2, \ldots, a_{n-1}$ are distinct. They are all smaller than n, since each is the smallest positive number not contained among at most $n-2$ options; therefore, they are a permutation of $\{1, 2, \ldots, n-1\}$.

Second, consider b_k for $1 \leq k \leq n-2$. Since none of $a_1, a_2, \ldots, a_k$ can equal b_k, but b_k is an integer from 1 to n, we know that it must be an element of $\{a_{k+1}, \ldots, a_{n-1}, n\}$. From this observation, it follows that we can define a graph T_k with $V(T_k) = \{a_k, a_{k+1}, \ldots, a_{n-1}, n\}$ and $E(T_k) = \{a_k b_k, a_{k+1} b_{k+1}, \ldots, a_{n-1} b_{n-1}\}$: each edge in $E(T_k)$ really does have both endpoints in $V(T_k)$.

We are now ready to proceed with the proof. The graph T_k is not just any graph: it is a tree in which the leaf with the smallest number is a_k. We will prove this by an induction that starts with T_{n-1} and ends with T_1.

In the base case, T_{n-1} is the graph with vertices $\{a_{n-1}, n\}$ and edge $a_{n-1} b_{n-1}$; since $b_{n-1} = n$, this is an edge between T_{n-1}'s two vertices, so T_{n-1} is a tree. Both a_{n-1} and b_{n-1} are leaves, but since $b_{n-1} = n$ and $a_{n-1} \neq b_{n-1}$, a_{n-1} must be the smaller leaf.

Next, for some positive $k < n-1$, suppose T_{k+1} is a tree. The only difference between T_k and T_{k+1} is that we add vertex a_k and edge $a_k b_k$. By our first observation, $a_k \notin V(T_{k+1})$, so a_k is a leaf of T_k. Since T_{k+1} is a tree, it has no cycles, so T_k cannot contain any cycles not using vertex a_k. It cannot have any cycles using a_k, either, because a_k has degree 1. Therefore T_k still has no cycles; we know it has $n-k+1$ vertices and $n-k$ edges, so it is a tree by condition 5 of Theorem 10.2.

Each a_i for $i < k$ is not yet a vertex of T_k, by our first observation. Each b_i for $i \geq k$ appears a second time as a_j for $j > i$, by our second observation; so it is the endpoint of at least two edges of T_k, and is not a leaf. Among the remaining positive integers, a_k is the smallest; therefore in particular it is the smallest leaf of T_k. This completes the induction.

From this claim, it follows that the deletion sequence algorithm, when encountering tree T_k, will write down the pair (a_k, b_k) and delete a_k, then either go on to tree T_{k+1} or (if $k = n-1$) stop. In particular, if we start with tree T_1, the deletion sequence algorithm will proceed through the trees $T_2, T_3, \ldots, T_{n-1}$ and write down the sequence

$$(a_1, b_1), (a_2, b_2), \ldots, (a_{n-1}, b_{n-1}),$$

which is exactly what we wanted to show. □

With this setup complete, we are ready to finish counting. This theorem is known as Cayley's formula after Arthur Cayley, whom we already know as the mathematician that came up with the term "tree". (So, you see, Prüfer's argument was not the first to be found. However, as it sometimes happens, Cayley was not the first, either; the formula was first proven by Carl Wilhelm Borchardt in 1860 [9].)

Theorem 11.3. *There are n^{n-2} trees with vertex set $\{1, 2, \ldots, n\}$.*

Proof. Each tree with vertex set $\{1, 2, \ldots, n\}$ has a Prüfer code $(b_1, b_2, \ldots, b_{n-2})$ whose elements are integers between 1 and n, uniquely defined by the deletion sequence algorithm. Therefore the number of trees with vertex set $\{1, 2, \ldots, n\}$ is exactly the number of possible Prüfer codes.

Moreover, for every such sequence $(b_1, b_2, \ldots, b_{n-2})$, we can apply (11.1) to determine a_1, then a_2, and so on through a_{n-1}, as long as we go in that order: each a_k will be the smallest positive integer not contained in a set we've already entirely determined. We can also set $b_{n-1} = n$. Now, the sequence

$$(a_1, b_1), (a_2, b_2), \ldots, (a_{n-1}, b_{n-1})$$

is a deletion sequence of a tree with vertex set $\{1, 2, \ldots, n\}$, by Lemma 11.2, and therefore $(b_1, b_2, \ldots, b_{n-2})$ is the Prüfer code of that tree. This shows that every sequence $n-2$ integers from 1 to n is a valid Prüfer code. There are exactly n^{n-2} such sequences, since there are n options for each of the numbers b_1 through b_{n-2}, so the number of trees with vertex set $\{1, 2, \ldots, n\}$ is also n^{n-2}. □

11.4 Working with Prüfer codes

There are a few more difficult questions to ask about Prüfer codes, but let's first pause to make sure that we can return from the abstract results to concrete claims. Take a Prüfer code like $(2, 1, 2, 5, 4)$. How do we turn it back into a tree?

Let's write down what we know about the deletion sequence of that tree, even if there's still some blanks to be filled in:

$$(_, 2), (_, 1), (_, 2), (_, 5), (_, 4), (_, _).$$

A Prüfer code is always a sequence of $n-2$ numbers from 1 to n; since we started with 5 numbers, their values will range from 1 to 7. We fill in the blanks from left to right.

For the first blank, which is a_1, we use (11.1). There are no a-terms before a_1; however, a_1 needs to be distinct from $\{b_1, b_2, b_3, b_4, b_5\} = \{1, 2, 4, 5\}$. The smallest integer not on this list is 3, so we set $a_1 = 3$:

$$(3, 2), (_, 1), (_, 2), (_, 5), (_, 4), (_, _).$$

We continue to use (11.1) for the next blank, which is a_2. Here, the excluded values are a_1, b_2, b_3, b_4, b_5 or $3, 1, 2, 5, 4$. We fill in the blank with the first integer not on this list, which is 6:

$$(3, 2), (6, 1), (_, 2), (_, 5), (_, 4), (_, _).$$

We go on in this way. Some people prefer to arrange the entries in a $2 \times (n-1)$ table, where each a_i entry (in the first, initially empty row) must be distinct from everything to its left, as well as everything below it and to the right:

a_1	a_2	a_3	a_4	a_5	a_6
b_1	b_2	b_3	b_4	b_5	b_6

$=$

3	6	?			
2	1	2	5	4	

Regardless, we fill in $a_3 = 1$ (the smallest value not among $3, 6, 2, 5, 4$), then $a_4 = 2$ (the smallest value not among $3, 6, 1, 5, 4$), then $a_5 = 5$ (the smallest value not among $3, 6, 1, 2, 4$), then $a_6 = 4$ (the smallest value not among $3, 6, 1, 2, 5$). All this is done using (11.1); then, the last blank is $b_6 = 7$ by (11.2). The completed deletion sequence is

$$(3, 2), (6, 1), (1, 2), (2, 5), (5, 4), (4, 7).$$

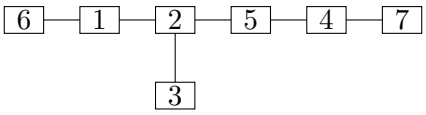


Figure 11.4: The tree with Prüfer code $(2, 1, 2, 5, 4)$.

This sequence tells us the edges of the tree, and if we like, we can draw a diagram such as the one in Figure 11.4.

That being said, it is not too often that we find ourselves needing to actually convert a Prüfer code back into a tree: it only matters for the proof of Theorem 11.3 that we can, in principle, do it.

This does not mean that Prüfer codes have no use beyond the proof of Theorem 11.3. The Prüfer code of a tree actually contains a bit of information about the tree, which we can use to solve more complicated counting problems. For example:

Proposition 11.4. *In the Prüfer code of a tree where* $\deg(x) = k$*, the number* x *appears* $k - 1$ *times.*

Proof. When we fill in the blanks in the sequence

$$(__, b_1), (__, b_2), \ldots, (__, b_{n-2}), (__, __)$$

we use each of the numbers $1, 2, \ldots, n$ once. The number n is going to fill in the second blank of the last pair, and the numbers in the first blanks are a permutation of $1, 2, \ldots, n-1$.

Therefore if a number x appears $k-1$ times in the Prüfer code, it appears k times in the deletion sequence

$$(a_1, b_1), (a_2, b_2), \ldots, (a_{n-1}, b_{n-1}).$$

But the deletion sequence is just a particular way to write down the edges of the tree, so if x appears in the deletion sequence k times, then it is the endpoint of k edges, which is just another way of saying that $\deg(x) = k$. □

For example, even before we turned the Prüfer code $(2, 1, 2, 5, 4)$ back into the tree shown in Figure 11.4, we could have known the rough structure of the tree:

- Vertices 3, 6, and 7 are leaves, because they do not appear in the Prüfer code at all.
- Vertices $1, 4, 5$ appear once, and therefore have degree 2.
- Vertex 2 appears twice, and therefore has degree 3.

Question:	What can we say about the shape of such a tree, knowing only the degrees of the vertices?
Answer:	Such a tree must consist of three paths starting at vertices 3, 6, 7 and converging at vertex 2.

Here is a quick example of using Proposition 11.4 to solve a counting problem:

(a) An n-vertex path graph

(b) An n-vertex star graph

Figure 11.5: Two trees with very different structures

Corollary 11.5. *There are $(n-1)^{n-2}$ trees with vertex set $\{1, 2, \ldots, n\}$ in which vertex 1 is a leaf.*

Proof. By Proposition 11.4, vertex 1 is a leaf (has degree 1) if and only if the number 1 never appears in the Prüfer code. There are $(n-1)^{n-2}$ such codes: the code is a sequence with $n-2$ terms, each of which is now restricted to one of the $n-1$ values in the set $\{2, 3, \ldots, n\}$. Therefore there are $(n-1)^{n-2}$ such trees. □

11.5 Unlabeled trees

Now that we've counted labeled trees on n vertices, we can try to say something about unlabeled trees. How many n-vertex trees are there, up to isomorphism? As I mentioned earlier in this chapter, we can think of this as counting equivalence classes of the n^{n-2} trees with vertex set $\{1, 2, \ldots, n\}$. Each equivalence class is a set of trees that are isomorphic (but not equal).

Sometimes, if we're very lucky, the equivalence classes all have the same size. In that case, to count the equivalence classes, we can divided by the number of elements of one equivalence class. Let's begin by destroying all such hopes: when we count trees, the equivalence classes are far from equal in size.

Question:	Figure 11.5a shows the path graph P_n. How many trees with vertex set $\{1, 2, \ldots, n\}$ are isomorphic to it?
Answer:	There are $n!$ ways to put the vertices in order from left to right along the path. However, the graph does not know a "left" and a "right": if you reverse the sequence, you get the same graph, drawn in reverse. Therefore there are $\frac{1}{2}n!$ paths with vertex set $\{1, 2, \ldots, n\}$.

Question:	Figure 11.5b shows the star graph S_n. How many trees with vertex set $\{1, 2, \ldots, n\}$ are isomorphic to it?
Answer:	As soon as we choose which vertex in the set $\{1, 2, \ldots, n\}$ is the vertex of degree $n-1$ at the center of the star, the graph is completely determined. Therefore there are n stars with vertex set $\{1, 2, \ldots, n\}$.

If all n-vertex trees were like P_n, then every equivalence class would have $\frac{1}{2}n!$ elements, and the number of equivalence classes would be the quotient $n^{n-2}/(\frac{1}{2}n!)$. If instead all n-vertex trees were like S_n, then every equivalence class would have n elements, and the number of equivalence classes would be $\frac{n^{n-2}}{n}$ or n^{n-3}.

In reality, neither of these extremes is the case. The truth is somewhere in the middle; but it is more like the first answer than the second. Why? Well, the reason that there are very few different trees isomorphic to S_n is that the star graph has a lot of symmetry. Most large trees are not nearly as symmetric, so most equivalence classes are pretty large.

A result known as Stirling's formula says that, very approximately, $n!$ grows like $(\frac{n}{e})^n$, where $e \approx 2.718$ is Euler's number.[18] If we plug this into the quotient $n^{n-2}/(\frac{1}{2}n!)$, we get an estimate of $2e^n/n^2$ for the number of unlabeled trees. This is not the true growth rate, but it is the right type of growth: the number of unlabeled trees really does grow exponentially. That growth rate was first precisely analyzed in 1937 by George Pólya [83], who described it as an exponential c^n with $c \approx 2.9557$, divided by a polynomial factor.

No exact formula is known. In the Online Encyclopedia of Integer Sequences, the number of n-vertex unlabeled trees can be found in one of the very first sequences: sequence A000055 [78]. The first few terms are

$$1, 1, 1, 1, 2, 3, 6, 11, 23, 47, 106, 235, \ldots$$

The initial 1's in this sequence correspond to $n = 0$[19] through $n = 3$, where only one n-vertex tree is possible. (For $n = 4$, we have two options: the 4-vertex path, and the 4-vertex star.)

11.6 Practice problems

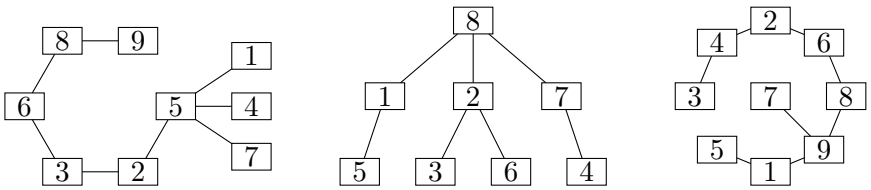
1. Find the trees with the following Prüfer codes:

   a) $(3, 3, 3, 3, 3)$.

   b) $(2, 7, 1, 5, 4, 6)$.

   c) $(3, 1, 4, 1, 5, 9, 2)$.

2. Find the Prüfer codes of the following trees:



   (One of these Prüfer codes gives you my birth date. Another is the first few digits of Euler's number e. Another tells you the phone number to call to reach Ghostbusters.)

[18] See the second problem at the end of this chapter if you want to know more digits of this constant, but prepare to do some work, first.

[19] I actually disagree with the OEIS on the initial value; I do not believe there is a 0-vertex tree. Even if we allow 0-vertex graphs to be exist, they would surely have 0 edges, but an n-vertex tree should have $n - 1$ edges. My opinion, which does not really affect anything but definitions and initial terms, is that the 0-vertex graph exists but is not connected.

3. Find all 16 trees with vertices $\{1, 2, 3, 4\}$.

4. What is the Prüfer code of the path graph in Figure 11.5a, and what is the general form of a Prüfer code for a tree isomorphic to it?

5. What is the Prüfer code of the star graph in Figure 11.5b, and what is the general form of a Prüfer code for a tree isomorphic to it?

6. Use Prüfer codes and Proposition 11.4 to count:

   a) The number of trees with vertex set $\{1, 2, \ldots, n\}$ in which vertex 1 has degree 3.

   b) The number of trees with vertex set $\{1, 2, \ldots, n\}$ in which vertex 1 has degree $n - 2$.

   c) The number of trees with vertex set $\{1, 2, \ldots, n\}$ in which all vertices except vertex 1 and 2 have degree 1.

   For parts (b) and (c), think about how you would count them without using Prüfer codes.

7. Use Corollary 11.5 to find the average number of leaves in an n-vertex tree.

8. Prove that if a tree has maximum degree d, then it has at least d leaves:

   a) Using Prüfer codes and Proposition 11.4.

   b) Using ideas from Chapter 10.

9. Is it true that if only one number is changed in a Prüfer code, then this only changes the tree it corresponds to by one edge?

   If so, prove it. If not, find (in terms of n) the maximum number of edges that can change in the tree, if the Prüfer code changes in only one position.

10. Let $f(n)$ be the number of trees with vertex set $\{1, 2, \ldots, n\}$ that contain the edge 12.

    a) Prove that $f(n)$ is also the number of trees with vertex set $\{1, 2, \ldots, n\}$ that contain the edge xy, for any other edge xy that such a tree could have.

    b) Using part (a), find and prove a formula for $f(n)$.

    c) Let G be a graph with n vertices and $\binom{n}{2} - 1$ edges. In terms of n, find the number of spanning trees G has.

12 Directed acyclic graphs

The purpose of this chapter

It may be a slightly odd choice to put a chapter on directed acyclic graphs in the part of the book about trees, but I think there is a thematic fit. We will see some analogies between directed acyclic graphs and trees. (Or should it be forests?)

Additionally, just as trees can help us study connected graphs, directed acyclic graphs can help us study the connectedness of digraphs, via strongly connected components and condensations. This is mostly a coincidence, though; the use of directed acyclic graphs is very different from the use of trees. However, in this way, we can take a second look at some of the concepts from Chapter 3, and how they change when the graph is directed.

Since we'll be dealing exclusively with directed graphs for the entire chapter, be sure to keep the definitions and results from the second half of Chapter 7 in mind as you read.

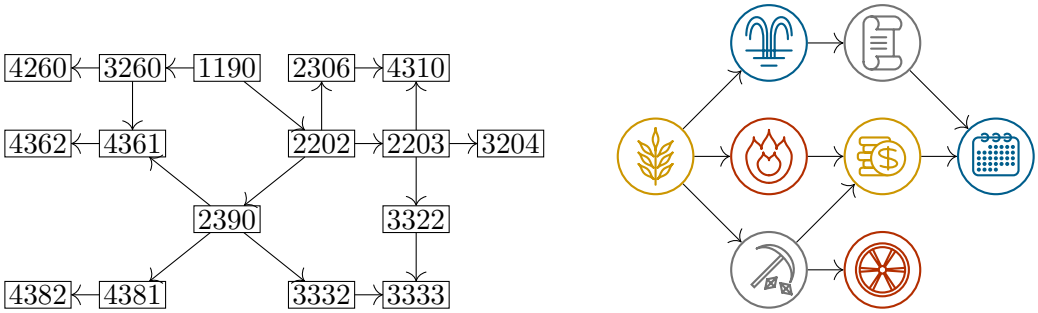
12.1 Directed acyclic graphs

Figure 12.1a shows one example of the kind of graph we will study in this chapter—loosely inspired by real life. It represents the course prerequisite relationship between several math courses offered at Kennesaw State University. It also includes one fictional course: MATH 3333, which is what I imagine the course number for a course on random graphs would be. (MATH 3333 doesn't really exist, but I have often thought that if it did, it would have been so fun to teach.)

With or without this fictional innovation, there is one important property that the digraph in Figure 12.1a should have, from the point of view of graph theory. Unless something is very wrong, it should not have a cycle!

Question:	What would go wrong if we had a cycle? For example, what would go wrong if we added an arc $(3322, 2202)$?
Answer:	There would be no way to take the courses in the cycle while obeying the prerequisites. Each course in the set $\{2202, 2203, 3322\}$ has a prerequisite in that set, giving no starting point.

This is such a common restriction (equally applicable to course offerings, baking recipes, or assembly lines) that the directed graphs obeying it have their own name.



(a) Do not use as academic advice

(b) An example of a research tree

Figure 12.1: Two examples of directed acyclic graphs

Definition 12.1. *An **directed acyclic graph**, or sometimes "dag" for short, is a directed graph that contains no cycles.*

Figure 12.1b shows another example of a directed acyclic graph: my own simplified representation of a research tree from a game like *Civilization*. This one also represents prerequisites: this time, for acquiring new technologies in the game. We imagine that, for example, until a civilization masters the use of writing, it cannot possibly hope to invent the calendar: hence the arc from the "writing" node to the "calendar" node.

Although this structure is called a research tree, it doesn't exactly resemble the undirected graphs we call trees in every way. For one, we don't require the digraph to be connected. (If we also included all the German classes at KSU in Figure 12.1a, we would probably not see any arcs between the two types of classes.) This means that there is no lower bound on the number of arcs; we should be comparing directed acyclic graphs to forests, not trees. However, the upper bound on the number of arcs is much weaker as well.

Question:	Suppose you can alter the course of history and change the relationships between the 8 technologies in Figure 12.1b to any directed acyclic graph you like. What is the maximum number of arcs you can make?
Answer:	If the only requirement is to avoid cycles, we can have up to 28 arcs, by arranging the 8 technologies in a line and drawing arcs from each technology to all those that come after it.

This 28-arc solution has some redundant arcs: if arcs (x, y) and (y, z) exist, then as far as modeling prerequisites goes, there is no need for arc (x, z). But redundancy can't take all the blame.

Question:	What is the maximum number of arcs if we never include arc (x, z) as long as both (x, y) and (y, z) exist?
Answer:	We can still get up to 16 arcs by dividing the technologies into four "basic" and four "advanced" ones, and adding an arc from each basic technology to each advanced technology.

There are some parallels between directed acyclic graphs and trees, though. In trees, we have learned to look for leaves: vertices of degree 1. In directed acyclic graphs, we instead look for *sources* (vertices of indegree 0, as defined in Chapter 7) and *sinks* (vertices of outdegree 0).

Lemma 12.1. *Every directed acyclic graph has at least one source and at least one sink.*

Proof. As far as the existence of a source goes, this is exactly the contrapositive of Theorem 7.2, and the existence of a sink is the contrapositive of a "mirror version" of that theorem. But we can also prove this result from scratch.

Let D be a directed acyclic graph, and let the walk $(x_0, x_1, \ldots, x_l)$ represent the longest path in D. There cannot be an arc (y, x_0) for any $y \in V(D)$: if y is a vertex on the path, then a cycle is created, and if not, then the walk $(y, x_0, x_1, \ldots, x_l)$ represents a longer path. Therefore x_0 is a source.

Similarly, there cannot be an arc (x_l, z) for any $z \in V(D)$: if z is a vertex on the path, then a cycle is created, and if not, then the walk $(x_0, x_1, \ldots, x_l, z)$ represents a longer path. Therefore x_l is a sink. □

The parallel is not only in the proof technique (which, by the way, could be directly adapted to a proof of Lemma 10.5 on the existence of leaves in a tree). Just like leaves, sources and sinks are the best choice when writing a proof by induction on directed acyclic graphs, or when coming up with a recursive algorithm. Although it's true that for every vertex x in a directed acyclic graph D, the digraph $D - x$ also has no cycles, it is still best to remove a source or sink, because they're often easiest to put back.

To give an example, suppose we are given a directed acyclic graph D, and want to compute the length of a longest path in D. To do so, we'll set ourselves a slightly harder task: for every vertex $x \in V(D)$, we will compute the value $f(x)$ equal to the length of the longest path in D ending at x. (If we want the absolute longest path, simply find the largest value of f over all vertices.)

If the directed acyclic graph D has only one vertex, we will assign that vertex a value of 0, because D cannot have a path of any positive length. If D has more vertices, then we compute f recursively: let x be any sink in D, and apply our algorithm to $D - x$.

In D, for any vertex $y \neq x$, the value $f(y)$ should be the same as in $D - x$. There cannot be any path ending at y that goes through x, because it's impossible for a path to leave x. So we only need to evaluate $f(x)$. To do this, let $y_1, y_2, \ldots, y_k$ be the vertices with an arc to x: then every path to x is either the length-0 path that starts at x, or enters x via some y_i, so we define

$$f(x) := \max\{0, f(y_1), f(y_2), \ldots, f(y_k)\}.$$

Now we have computed f for every vertex of D.

Question:	If all we needed was a longest path in D, and we did not care where it ended, why did we decide to compute $f(x)$?
Answer:	If we want to work recursively, but only compute longest paths, we might miss the longest path in D if it didn't extend a longest path in a subgraph of D.

Question:	Why is it necessary for the removed vertex x to be a sink?
Answer:	If x is not a sink, then in going from $D - x$ to D, we might have to change many other f-values in addition to $f(x)$.

12.2 Topological orders

Suppose you buy a piece of furniture which comes disassembled with a set of instructions for how to put it together. In principle, the necessary steps could form an arbitrary directed acyclic graph, and there is some freedom in deciding which steps to perform before which others. However, the instructions probably list the steps in order, to avoid confusion.

Given a directed acyclic graph D, a **topological order** of D is a generalization of this idea. It is a sequence $x_1, x_2, \ldots, x_n$ listing all the vertices of D such that for every arc $(x_i, x_j) \in E(D)$, vertex x_i comes before vertex x_j in the topological order: $i < j$. For example, given the class prerequisites shown in Figure 12.1a, one possible topological order is

$$1190, 2202, 2203, 2306, 2390, 3260, 3204, 3322, 3332, 3333, 4260, 4310, 4361, 4362, 4381, 4382$$

which puts the courses in numerical order. (I assume that most departments in most universities make sure that the numerical order on their courses is also a topological order, though it is not always the only order or the best order in which to take the courses.)

Proposition 12.2. *Every directed acyclic graph D has a topological order.*

Proof. We induct on the vertices of D. If D has only one vertex, then there is only way to list the vertices in a sequence, and it is vacuously a topological order.

Let $n > 1$; suppose D has n vertices, and that every $(n-1)$-vertex directed acyclic graph has a topological order. By Lemma 12.1, D has a sink x. By the inductive hypothesis, $D - x$ has a topological order $x_1, x_2, \ldots, x_{n-1}$. Let $x_n = x$. Then for every arc (x_i, x_j), we consider three cases:

- If $i < n$ and $j < n$, then (x_i, x_j) is an arc of $D - x$, and since we chose $x_1, x_2, \ldots, x_{n-1}$ to be a topological order of $D - x$, we must have $i < j$.
- If $i < n$ and $j = n$, then $i < j$ by substitution, with no need for graph theory.
- The case $i = n$ and $j < n$ is impossible, because $x_n = x$ is a sink, and there are no arcs out of it.

In all possible cases, $i < j$, and so $x_1, x_2, \ldots, x_n$ is a topological order of D. By induction, topological orders exist for directed acyclic graphs with any number of vertices. □

Question:	Can a directed graph D have a topological order if it has a cycle?
Answer:	No: if D has a topological order $x_1, x_2, \ldots, x_n$, then no path that leaves x_i can return it, because all arcs only take you forward in the topological order. Therefore D cannot contain any cycles.

Proposition 12.2 is a good illustration of how we can use Lemma 12.1 to prove properties of directed acyclic graphs by induction. However, in many cases, it also makes induction (and recursive algorithms) unnecessary. For example, the algorithm in our previous section for computing the longest directed graph can now be rephrased more directly. First, choose a topological order $x_1, x_2, \ldots, x_n$. Then, for each $i = 1, 2, \ldots, n$, compute $f(x_i)$ by using the values of some of the previous vertices $f(x_1), f(x_2), \ldots, f(x_{i-1})$.

12.3 Strongly connected

We will now abandon directed acyclic graphs for a moment and turn to a seemingly unrelated topic: the question of what it means for a directed graph to be connected.

In Chapter 8, we had to develop one definition of connectedness for directed graphs. We defined a *weakly connected* digraph to be a digraph whose underlying graph is connected. This is not always the wrong definition, and in fact, we developed it with the best intentions possible: it was the definition we needed! However, I also find it vaguely unsatisfying, because it doesn't tell us anything about directed walks at all.

The opposite approach is to directly copy the definition of connectedness. We will say that a directed graph D is *strongly connected* if, for every $x \in V(D)$ and every $y \in V(D)$, there is a directed $x - y$ walk. Because we could have chosen y, x instead of x, y, there must also be a directed $y - x$ walk. (For directed graphs that are not strongly connected, this is not guaranteed to be true.)

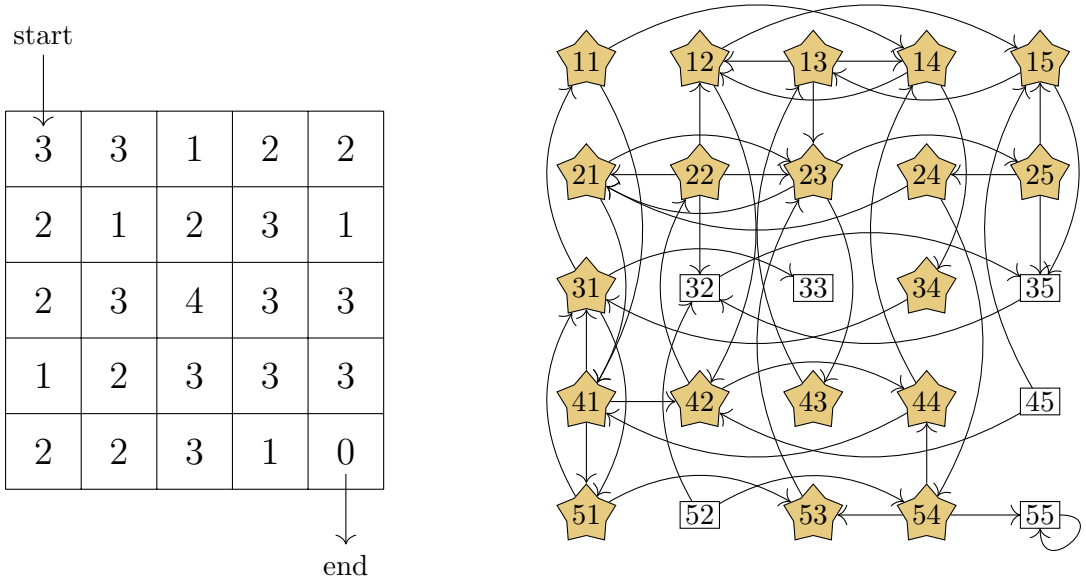
Question:	Is a directed path graph $\overrightarrow{P_n}$ strongly connected? What about a directed cycle graph $\overrightarrow{C_n}$?
Answer:	The directed path graph is not strongly connected for $n > 1$: in fact, it's a directed acyclic graph! You can never get to an earlier vertex from a later vertex. Meanwhile, $\overrightarrow{C_n}$ is strongly connected: for any x and any y, if you follow the cycle starting at x, you will eventually reach y.

It's useful to observe that Theorem 3.1 (and its proof) continue to work for directed graphs: whenever there is an $x - y$ walk, there is also a $x - y$ path.

We used connectedness in undirected graphs to define connected components, and we could use strong connectedness to define strongly connected components in the same way: using equivalence relations. Since we have a second chance to look at such a problem, though, let's take that chance to arrive at our destination differently.

We will define a *strongly connected component* of a digraph D to be a maximal strongly connected subgraph of D: a subgraph D' which is strongly connected, and which is not a subgraph of any other strongly connected subgraph of D. In other words, we cannot add any more vertices or arcs of D to D' and still get a strongly connected result.

Figure 12.2 shows an example of a strongly connected component in the same numerical maze that we used as an example in Chapter 7. The subgraph induced by the marked vertices in



(a) A numerical maze

(b) A strongly connected component

Figure 12.2: Strongly connected components in a numerical maze

Figure 12.2b is a large strongly connected component in the directed graph. From any marked vertex, it is possible to reach any other; the other vertices either cannot be reached from the marked vertices, or cannot reach any marked vertex.

I found the strongly connected component by starting with just vertex 11 and looking for cycles. Whenever I found a cycle that contained a marked vertex, I also marked every other vertex of the cycle.

Question:	Why does this guarantee that the result is a strongly connected component?
Answer:	From every vertex on the cycle, we can reach the marked vertex on the cycle, and from there, we already know we can reach every previously-marked vertex. This strategy also works in reverse.

Compared to what we did in Chapter 3, we've definitely done less work to define what a strongly connected component is. However, now we must do more work to ensure that the definition behaves:

Theorem 12.3. *If $D_1, D_2, \ldots, D_k$ are the strongly connected components of a directed graph D, then $V(D_1), V(D_2), \ldots, V(D_k)$ are a partition of $V(D)$.*

Proof. When we say that $V(D_1), V(D_2), \ldots, V(D_k)$ are a partition of $V(D)$, we say two things: first, that every $x \in V(D)$ is in $V(D_i)$ for some i, and second, that no $x \in V(D)$ is in both $V(D_i)$ and $V(D_j)$ where $i \neq j$.

For the first claim, to find a strongly connected component containing a vertex x, start with the subgraph consisting of x alone.

Question:	Why is this subgraph strongly connected?
Answer:	There is only one thing to check: that there is an $x - x$ walk. The sequence (x) is an $x - x$ walk of length 0.

Either this subgraph is a strongly connected component, or else (by the contrapositive of that definition) it must be contained in a bigger strongly connected subgraph. That subgraph, in turn, is either a strongly connected component, or contained in a bigger strongly connected subgraph. We can skip to the end by applying the extremal principle: take a strongly connected subgraph containing x which has as many vertices and arcs as possible. From the way we chose it, we know that it cannot be contained in a larger strongly connected subgraph, so it satisfies the definition of a strongly connected component.

For the second claim, suppose for the sake of contradiction that $x \in V(D_i) \cap V(D_j)$ for some $i \neq j$. The contradiction is this: the union of directed graphs $D_i \cup D_j$ is a bigger strongly connected subgraph of D. To see this, take any $y \in V(D_i \cup D_j)$ and $z \in V(D_i \cup D_j)$. Then there is a $y - x$ walk: if $y \in V(D_i)$, this is true because $x \in V(D_i)$ and D_i is strongly connected, and if $y \in V(D_j)$, this is true because $x \in V(D_j)$ and D_j is strongly connected. There is also an $x - z$ walk, for the same reason.

Joining these walks together, we get a $y - z$ walk, proving that $D_i \cup D_j$ is strongly connected. But then D_i and D_j are both subgraphs of $D_i \cup D_j$, which is a strongly connected subgraph of D, so they cannot be strongly connected components. This contradiction tells us that x cannot be in two different strongly connected components, proving the theorem. □

The proof of Theorem 12.3 is a special case of the general proof that, given an equivalence relation on a set S, its equivalence classes form a partition of S. If you look closely, you can see where elements of this proof rely on the three defining properties of an equivalence relation.

Be careful, however: it is not true that D is equal to $D_1 \cup D_2 \cup \cdots \cup D_k$, the union of its strongly connected components! As we are about to see, there are other arcs D might have.

12.4 Condensations

When we worked with undirected graphs, the connected components told us the whole story of which vertices can reach which other vertices: there is an $x - y$ walk in G if and only if x and y are in the same connected component of G. With directed graphs and strongly connected components, this is no longer true.

Question:	Why are strongly connected components different?
Answer:	If x and y are in different strongly connected components, there can still be either an $x - y$ walk or a $y - x$ walk; just not both.

We can, however, guarantee at least one thing: if we know the relationship between two vertices x and y, then the same relationship holds between every vertex x' in x's strongly connected

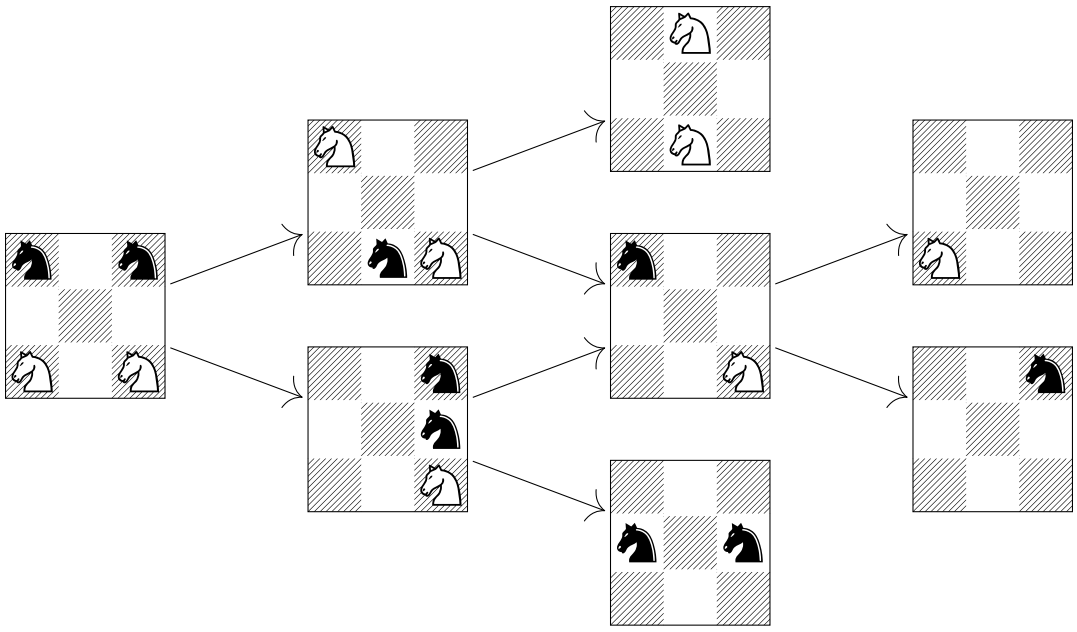


Figure 12.3: The condensation of the state digraph of the four knights puzzle

component and every vertex y' in y's strongly connected component. For example, if there is an $x - y$ walk, then there is also an $x' - y'$ walk: we go from x' to x to y to y'.

For this reason, once we find the strongly connected components $D_1, D_2, \ldots, D_k$ of a digraph D, we can summarize the relationship between different strongly connected components much more concisely. We define a new directed graph called the *condensation* of D to be the directed graph whose vertices are $D_1, D_2, \ldots, D_k$, with an arc (D_i, D_j) if $i \neq j$ and there is an $x-y$ walk where $x \in V(D_i)$ and $y \in V(D_j)$.

To give an example of this, I will add a frankly unnecessary twist to a classic puzzle. In this puzzle, two white knights and two black knights are placed on a 3×3 chessboard, as shown in the leftmost vertex of Figure 12.3. The knights move as in ordinary chess: two squares in one direction and one square in a perpendicular direction. The objective of the puzzle is to switch the positions of the black and white squares. The puzzle can be solved if black and white must take turns, but for the purposes of this example, I will not require this.

My unnecessary twist is to allow moves where a knight captures a knight of the opposite color. Obviously, once you do this, you have no hope of solving the puzzle, since you can't put back the captured piece. It also means that if we want to define a graph whose vertices are the possible states of the puzzle, it must be a directed graph: some moves cannot be undone.

Figure 12.3 doesn't show this state digraph, but rather, its condensation. Each 3×3 board shows a single vertex of its strongly connected component.

Question:	What are the strongly connected components of this digraph, then?
Answer:	For each possible collection of pieces we can have, there is a strongly connected component. By making moves that are not captures, we can rearrange the pieces however we like; a capture takes us to a different strongly connected component. For example, each of the rightmost 3×3 boards, with only one knight on it, represents a strongly connected component with 8 vertices, one for each possible position of the remaining knight.

You might notice something else about Figure 12.3: it is a directed acyclic graph. This is not an accident, but it is true in general.

Theorem 12.4. *The condensation of any directed graph is a directed acyclic graph.*

Proof. Let D be a directed graph, and suppose for the sake of contradiction that the condensation of D contains a cycle. Let $(D_0, D_1, \ldots, D_{l-1}, D_0)$ represent one such cycle; each D_i here is a vertex of the condensation, so it is a strongly connected component of D.

By definition of the condensation, for each $i = 0, 1, \ldots, l-1$, there is some $x_i \in V(D_i)$ and some $y_{i+1} \in V(D_{i+1})$ such that (x_i, y_{i+1}) is an arc of D; when $i = l-1$, there is some $y_0 \in V(D_0)$ such that (x_{l-1}, y_0) is an arc. Because each D_i is strongly connected, it has a $y_i - x_i$ path. Using the arcs (x_i, y_{i+1}) to join these paths together, we obtain a cycle C containing a vertex from multiple strongly connected components.

But then, the union $D_0 \cup D_1 \cup \cdots \cup D_{l-1}$ is strongly connected: if $z_i \in V(D_i)$ and $z_j \in V(D_j)$, there is a $z_i - x_i$ walk in D_i, an $x_i - y_j$ walk that follows cycle C, and a $y_j - z_j$ walk in $V(D_j)$, so there is a $z_i - z_j$ walk. This invalidates the assumption that the individual D_i's are strongly connected components, since they are contained in a bigger strongly connected subgraph. □

This result makes it easy to use the condensation as a reference to see which vertices in the original digraph can reach which other vertices. To see if there is an $x - y$ walk, look up the strongly connected components of x and y, and see if there is an $x - y$ walk between them in the condensation. Not only are we potentially reducing the size of the problem, we are also simplifying it, because now we are working with a directed acyclic graph.

Question:	What is the condensation of a directed acyclic graph D?
Answer:	It is essentially D itself: each strongly connected component of D is just a vertex. Once we follow an arc out of a vertex x, we can never return to x, because that would imply the existence of a cycle.

12.5 Practice problems

1. Draw the condensation of the graph in Figure 12.2.
2. Medieval alchemists valued above all the following seven materials:
   - Alkahest: the universal solvent, made by adding crushed Bluemyrtle to molten Firemetal.
   - Bluemyrtle: a plant that can only be harvested with tools made of Firemetal.
   - Calomel: a dry powder left after dissolving Gold in Alkahest.
   - Dreadwater: the infusion of Bluemyrtle in plain water.
   - Elixir: a potent potion obtained by adding Gold powder to Dreadwater.
   - Firemetal: a mystical transformation of Gold.
   - Gold: could be transmuted from common dirt by adding Alkahest and Dreadwater.

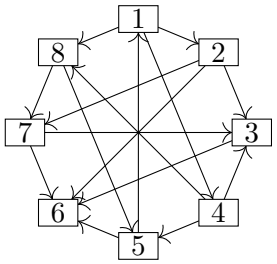
   Let H be the directed graph whose vertices are these compounds, with an arcs (x, y) representing that x is needed to synthesize y.

   a) Find a cycle in H, and explain why this would have been a problem for medieval alchemists.

   b) Describe the strongly connected components of H.

   c) Suppose that you are a very rich medieval alchemist with an unlimited supply of Gold. For you, the graph H must be modified to delete all arcs to this vertex, since you don't need anything else to synthesize it.

   Find a topological order of the modified graph.
3. In the directed graph shown below, find an arc (x, y) such that if it is reversed (deleted and replaced by (y, x)), the result is a directed acyclic graph. Give a topological order of the result.



4. If S is a set of integers, let the divisibility digraph of S be the directed acyclic graph with vertex set S and an arc (x, y) whenever $x \neq y$ and y is divisible by x. For example, if $S = \{2, 3, 6\}$, then the divisibility digraph of S has arc set $\{(2, 6), (3, 6)\}$.

   a) Draw a diagram of the divisibility digraph of $\{2, 3, 4, 6, 8, 9, 10, 12\}$.

   b) Prove that for every set S, the default order of the vertices (from smallest to largest) is a topological order of the divisibility digraph.

c) For the divisibility digraph in (a), find a different topological order: one in which 3 comes after 8.

d) Let D be the divisibility digraph of the set $\{1, 2, \ldots, 100\}$. Which vertex in this graph has indegree 1 and outdegree 32?

5. A game is played with two piles of stones. On a turn, a player picks a pile which is not empty, and takes one or more stones from it. The game ends once both piles are empty.

   We can represent this game by a digraph whose vertices are the states, with arcs representing the valid moves. Let $D_{n,m}$ be the digraph we get when the game is played with a pile of size n and a pile of size m.

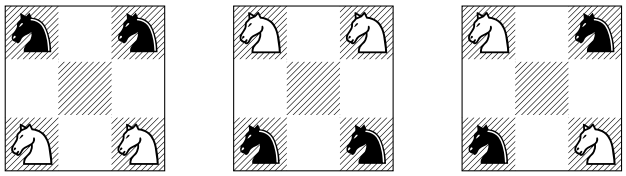
   a) Draw a diagram of $D_{3,2}$. (It should have 12 vertices.)

   b) Why is $D_{n,m}$ always a directed acyclic graph?

   c) Find a topological order of $D_{3,2}$. (There are many.)

6. We must be careful in the definition of the state digraph of the four knights puzzle if we want to get the condensation shown in Figure 12.3. Here are a few of the details.

   a) First of all, we must limit our vertices to the states that can be reached from the starting state. If not, then the leftmost vertex would have to be split into two vertices. To see why, prove that from the initial position shown below, we can reach the second (solving the classical puzzle) but cannot reach the third (showing the necessity for a second vertex in the condensation).



   b) Second, we must allow moves from either color of knight in any order. If we require black and white moves to alternate, then the most obvious effect is that Figure 12.3 would have to have 73 different sink vertices, rather than 4: one for each position with only pieces of one color left. (After all, at that point, no more moves are possible.) This is why I decided to allow multiple moves from the same side in a row.

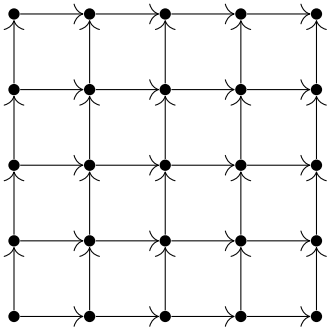
      Another interesting effect of requiring black and white moves to alternate is that one of the final states shown in the sink vertices of Figure 12.3 would no longer be reachable! Which one, and why not?

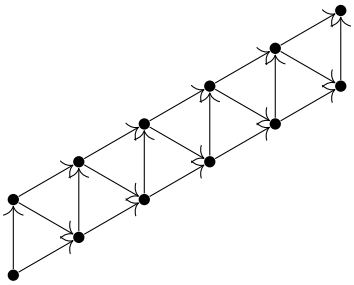
7. Call a vertex x in a directed graph D "safe" if it is impossible to leave x and get lost: for every vertex y such that D has an $x - y$ walk, D also has a $y - x$ walk.

   Prove that every directed graph has a safe vertex.

8. Let T be an n-vertex tree. Prove that there are exactly n directed acyclic graphs whose underlying graph is T.

9. a) A well-known brainteaser asks the following question: given a 5×5 grid, how many ways are there to go from the bottom left vertex to the top right vertex while only going up or to the right? We can phrase this in terms of directed graphs: how many paths are there from the bottom left vertex to the top right vertex of the first graph below?





a) Solve this brainteaser; see if you can generalize to $n \times n$ grids with a combinatorial formula.

b) How many paths are there from the bottom left vertex to the top right vertex in the second graph above? See if you can generalize this problem, as well.

c) How can this problem be solved efficiently for arbitrary directed acyclic graphs?

10. (AIME 2007) A frog is placed at the origin on the number line, and moves according to the following rule: in a given move, the frog advances to either the closest point with a greater integer coordinate that is a multiple of 3, or to the closest point with a greater integer coordinate that is a multiple of 13. A "move sequence" is a sequence of coordinates which correspond to valid moves, beginning with 0, and ending with 39. For example, $0, 3, 6, 13, 15, 26, 39$ is a move sequence. How many move sequences are possible for the frog?

Matchings

13 Bipartite matching

The purpose of this chapter

In this chapter, we will not yet see any of the big theorems about matchings. Here, we will only lay the foundations and begin to study bipartite graphs, matchings, and vertex covers.

I have gone to some trouble in this textbook to postpone the definition of bipartite graphs until this chapter. This is done for two reasons. First, I do not want the first few chapters to be overloaded with definitions and nothing but definitions—as much as I can help it, at least. Second, I do not want to give a definition when we have no use for it. This is because, to the extent I can, I want the definitions we make and the questions we ask about them to seem like reasonable definitions we make and reasonable questions to ask.

As a result, it is only now that I define bipartite graphs, because now we can ask the bipartite matching problem. Here, it is reasonable to ask whether (and in which way) a graph is bipartite, and so I hope that the definitions and initial theorems do not feel pointless.

13.1 Two chess puzzles

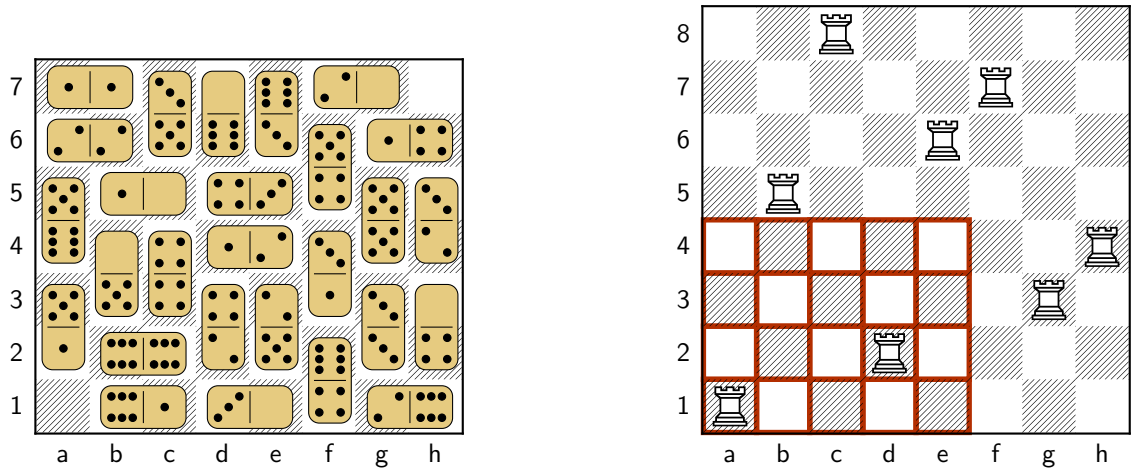
The title of this section may be a slight exaggeration; you will not need to know the rules of chess. These are, instead, two puzzles about placing pieces on a chessboard.

The first is a classic puzzle about placing dominoes on a chessboard. A complete set of dominoes has 28 dominoes; enough to completely cover an 8×7 grid. If I take away one domino, then it's possible to cover every square of the grid except for two opposite corners: Figure 13.1a shows one of many solutions.

But the standard chessboard is 8×8. So suppose you have 31 dominoes: a complete set with three extras. Can you cover every square of the 8×8 chessboard except for two opposite corners?

A second puzzle is shown in Figure 13.1b. Here, we place 8 rooks on a chessboard. A rook can move any number of spaces horizontally or vertically; we don't want the rooks to get in the way of each other, so we want to place them so that no two rooks are in the same row or column.[20] I'll add a condition to make this more difficult in a moment, but first...

[20] In the same "rank" or "file", if you're a chess purist.



(a) Dominoes on a 8×7 chessboard

(b) Rooks on a chessboard

Figure 13.1: Placing pieces on a chessboard

Question:	How many ways are there to place 8 rooks on an 8×8 chessboard so that no two rooks are in the same row or column?
Answer:	There are $8! = 40\,320$ ways. If we pick a column for the rook in row 1, then the rook in row 2, and so on, then we will have 8 options for the first rook, 7 for the second rook, 6 for the third rook, and so forth. By the product principle, we can multiply these numbers together, getting $8 \cdot 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$ or $8!$ arrangements.

Now I can ask you the real puzzle. Suppose we exclude the 4×5 area highlighted in Figure 13.1b. Is it still possible to place 8 rooks on the chessboard, still in different rows and columns, so that this area remains empty?

We will be able to answer both questions with the tools in this chapter. But first, let's take a moment to ask the question we've been asking since Chapter 1: how do we model each puzzle with a graph? In both cases, there are actually two reasonable approaches!

Let's start with the domino puzzle. We've already seen a tiling puzzle in Chapter 1, and we can model this puzzle in the same way. Create a graph with a vertex for each valid way to place a single domino: on an 8×8 chessboard with two opposite vertices removed, there will be 108 vertices. Then, add an edge between two vertices if they represent incompatible placements: dominoes that share a square of the board. To place 31 dominoes, we are looking for a 31-vertex *independent set*: a set of 31 vertices that do not have any edges between them.

As we will see in Chapter 18, finding independent sets is a hard problem. So let's instead take a different graph to start with: the 8×8 grid graph $G(8, 8)$, with vertices $(1, 1)$ and $(8, 8)$ removed.

Question:	If we model the chessboard with this graph, how do we interpret placing dominoes?
Answer:	Each placed domino corresponds to an edge of the graph. So we are looking for a way to select 31 edges that do not share any vertices.

Such a set of edges is called a matching. This part of the textbook is all about finding matchings! Please be patient, though; I will introduce matchings formally a bit later on in this chapter.

Now, let's move on to the rook puzzle. It, too, has two models: in one, we're looking for an independent set, and in the other, we're looking for a matching. We'll want to use the second model to solve the problem!

Question:	How do we model the rook puzzle with a graph so that a valid placement of rooks is an independent set?
Answer:	For this, let every square of the chessboard be a vertex (excluding the forbidden squares highlighted in Figure 13.1b, if we want to avoid them). Make two vertices adjacent if they share a row or column.

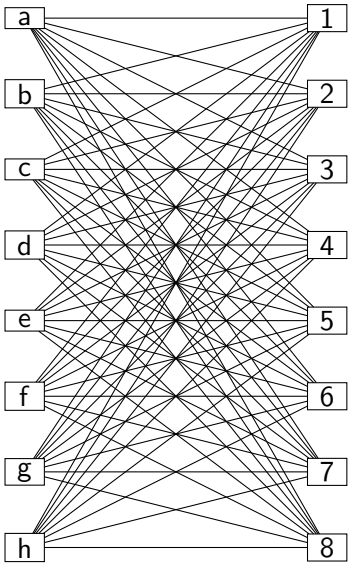
Question:	How do we model the rook puzzle with a graph so that a valid placement of rooks is a matching?
Answer:	For this, take a 16-vertex graph, with a vertex for each row or column. Make a row vertex adjacent to a column vertex if we allow a rook to be placed at the intersection of that row and column.

13.2 Bipartite graphs

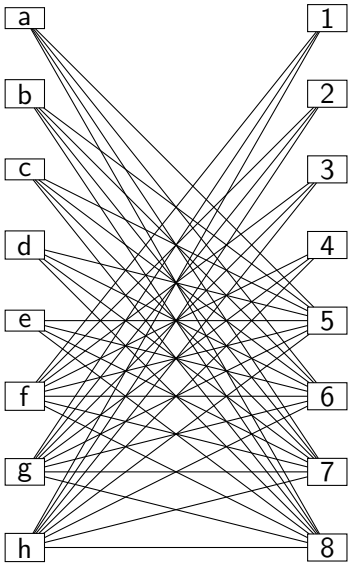
The graph we just defined for the rook placement problem has an unusual feature. Its vertices come in two types: the vertices representing rows, and the vertices representing columns. There is a special name for such graphs.

Definition 13.1. *A **bipartite graph** G is a graph for which we can choose disjoint sets A and B, with $A \cup B = V(G)$, such that every edge of G has one endpoint in A and one endpoint in B. The pair (A, B) is called a **bipartition** of G, and the sets A and B are called the two **sides**[21] of the bipartition.*

In the rook placement problem, we can take the bipartition (A, B) with $A = \{\mathsf{a}, \mathsf{b}, \ldots, \mathsf{h}\}$ (the columns of the chessboard) and $B = \{\mathsf{1}, \mathsf{2}, \ldots, \mathsf{8}\}$ (the rows). Figure 13.2 shows the two graphs we get in this way: Figure 13.2a shows the graph for placing rooks with no restrictions, and Figure 13.2b shows the subgraph when we forbid a 4×5 rectangle on the board.



(a) Graph for the entire chessboard



(b) Subgraph excluding the forbidden squares

Figure 13.2: Modeling rook placements with bipartite graphs

As always, we can draw a graph in any way we like; however, when we want to make it visually clear that a graph is bipartite it is traditional to arrange the two sides A and B in two columns (as in Figure 13.2) or in two rows.

The graph in Figure 13.2a representing the entire chessboard is a special case: it is isomorphic to a complete bipartite graph. (Just as the complete graph has all the edges a graph could possibly have, a complete bipartite graph has all the edges a bipartite graph could possibly have.) Here is one general definition:

Definition 13.2. *For any $m \geq 1$ and $n \geq 1$, the* ***complete bipartite graph $K_{m,n}$*** *is the graph with $m+n$ vertices $\{1, 2, \ldots, m+n\}$ and all mn possible edges xy where $1 \leq x \leq m$ and $m+1 \leq y \leq m+n$.*

We have already seen one special case: the star graph S_n is isomorphic to the complete bipartite graph $K_{1,n-1}$.

In the case of the graphs for the rook placement problem, the bipartition (A, B) is a natural part of the definition of the graph. The two vertices come in two types, representing the rows and the columns of the chessboard, and the rule defining adjacency is a relationship between the two types of vertex.

It turns out that the grid graph representing the domino problem is also a bipartite graph. Here, though, the bipartition (A, B) is not given to us. In order to see that the graph is bipartite, we have to find a pair (A, B) satisfying the definition of a bipartition.

[21] In graph theory literature, these have no consistent name; they can also be called "parts", or "blocks", or even "partite sets".

Question:	Can you think of a way to classify the squares of any chessboard into two types, so that every domino we place covers one square of each type?
Answer:	The classification is by color: let A be the set of light squares and let B be the set of dark squares.

The chessboard-style coloring proves that every grid graph $G(m, n)$ (and every subgraph of a grid graph) is bipartite. (If we remove opposite corners of the chessboard, we delete two vertices of the grid graph, but it remains bipartite when we do so.) Now, let's talk about how to find the pair (A, B) in general.

A problem that involves labeling the vertices of a graph (such as with two sides A and B) might be difficult because we might have to guess, make mistakes, and backtrack. In the case of a bipartition, we are actually quite lucky: there are frequent forced deductions. In a bipartite graph, if there is an edge xy where $x \in A$, then we know that $y \in B$ without guessing. The following algorithm relies only on making such forced deductions.

Given a graph G, we intend to give each vertex a label, A or B, to indicate which side of the bipartition it is on. Initially, all vertices start unlabeled.

1. Pick an arbitrary unlabeled vertex and label it with A.
2. Go through all vertices newly labeled with A, and label all their neighbors with B.
3. Go through all vertices newly labeled with B, and label all their neighbors with A.
4. Repeat steps 2–3 until a vertex has been labeled with both A and B, or until no more unlabeled vertices are being labeled.
5. If the graph is not connected, repeat steps 1–4 for each connected component.

If the graph we are working with is bipartite, then at the end, we get a bipartition (A, B), where A is the set of all vertices labeled with A, and B is the set of all vertices labeled with B.

Question:	What happens if the graph is not bipartite?
Answer:	We will give a vertex both labels, and stop. Such a vertex cannot exist in a bipartite graph, so if we are forced to have such a vertex, then the graph cannot be bipartite.

Question:	Why are we free to label a vertex with A in step 1: at the beginning, and when we consider each new connected component?
Answer:	In each connected component, we can swap the roles of A and B. So for every solution where this vertex is in B, there is another solution where it is in A.

The bipartition algorithm is quite similar to the distance-finding algorithm we studied in Chapter 3: in both algorithms, we visit all the vertices by a breadth-first search. The only difference is in what we do when we visit the vertices. In the distance-finding algorithm, we gave a vertex labels $0, 1, 2, 3, \ldots$ at successive stages. In this algorithm, we alternate the labels $A, B, A, B, \ldots$ at successive stages, instead.

Question:	Let G be a connected graph, and let x be the vertex initially placed in A. How can we define the sets A and B in terms of distances from x?
Answer:	In the end, A will be the set of all vertices at an even distance from x, and B will be the set of all vertices at an odd distance from x.

13.3 Odd cycles

Although the bipartition algorithm is a convenient way to test a specific graph for being bipartite, it is not as useful theoretically: when proving a general result, we would like some testable criteria. One way to prove a graph is bipartite is to find a bipartition (A, B) and show that it satisfies the definition, but we'd also like to be able to prove that a graph is not bipartite. For this, we have the following theorem:

Theorem 13.1. *A graph G is bipartite if and only if it contains no cycles of odd length (no odd cycles, for short).*

We can interpret this theorem as saying that the odd cycle graphs $C_3, C_5, C_7, C_9, \ldots$ are the simplest non-bipartite graphs: every other graph that isn't bipartite contains one of them.

Proof of Theorem 13.1. First, we prove that in a bipartite graph, all cycles must have even length. In fact, we will prove that all closed walks have even length! The length of a cycle is equal to the length of a walk representing it, so this will be sufficient.

Let G be bipartite with bipartition (A, B), and let $(x_0, x_1, \ldots, x_{l-1}, x_0)$ be a closed walk. Without loss of generality, suppose that $x_0 \in A$. Because edges x_0x_1, x_1x_2, and so on must have one endpoint on each side of the bipartition, we must have $x_1 \in B$, $x_2 \in A$, and so on. In general, we can prove by induction that $x_i \in A$ whenever i is even, and $x_i \in B$ whenever i is odd: the base case is x_0, and then induction step simply uses the fact that the endpoints of edge x_ix_{i+1} must be on opposite sides.

Since the endpoints of edge $x_{l-1}x_0$ must be on opposite sides, and $x_0 \in A$, we must have $x_{l-1} \in B$, which means $l - 1$ is odd. Therefore l, the length of the walk, must be even. The first half of the proof is complete!

It is easy at this point in the proof to get confused and prove the same thing we just did a second time—for example, this would happen if we continued by proving that if G contains an odd cycle, it is not bipartite. To continue correctly, we will assume that G is not bipartite, and prove that it contains an odd cycle.

The condition "G is not bipartite" is very difficult to work with: it says that a bipartition (A, B) does not exist, or that every candidate pair (A, B) somehow fails to be a bipartition. To make use of this, we define a pair (A, B) that ought to be a bipartition in any bipartite graph. Inspired by our bipartition algorithm, we begin by choosing a vertex from each connected component of G, and then define:

- A to be the set of all vertices at an even distance from a chosen vertex;
- B to be the complement of A: the set of all vertices at an odd distance from a chosen vertex.

Since G is not bipartite, this is not a bipartition. So there must be some edge xy between two vertices both in A, or both in B. Since x and y are necessarily in the same connected component, they must both be at an even distance, or both at an odd distance, from the chosen vertex z in that component.

Consider the walk that begins at x, follows some shortest $x - z$ walk, then follows some shortest $z - y$ walk, then takes the edge xy. The first two segments of this walk both have odd length or both have even length, so combined, their length is even. The final edge xy increases the length by 1. We've found a closed walk of odd length.

To complete the proof, let $(x_0, x_1, \ldots, x_{l-1}, x_0)$ be a shortest closed walk of odd length: we will prove that it represents an odd cycle. Suppose not: then there are some positions i and j with $0 \leq i < j < l$ and $x_i = x_j$.

Question:	The other way the closed walk can fail to represent a cycle is if $l < 3$. Why can we rule this out?
Answer:	Since l is odd, the only such case is $l = 1$. But a closed walk of length 1 cannot exist, because a vertex cannot be adjacent to itself. (In a multigraph, this would be possible with a loop, but in a multigraph, the loop would also be an odd cycle.)

The closed walk $(x_i, x_{i+1}, \ldots, x_j)$ has length $j - i$; the closed walk

$$(x_0, x_1, \ldots, x_i, x_{j+1}, x_{j+2}, \ldots, x_{l-1}, x_0)$$

has length $l - (j - i)$. Since the sum of these two lengths is the odd number l, at least one of these lengths must be odd; since $i < j < l$, both lengths are smaller than l. Therefore we've found an even shorter closed walk of odd length, contradicting our initial choice of a walk. We conclude that the pair i, j cannot exist: the closed walk we found does represent an odd cycle! This completes the proof. □

Question:	In the second half of the proof, why do we need to laboriously define A and B; can we just skip directly to defining $(x_0, x_1, \ldots, x_{l-1}, x_0)$ to be a shortest closed walk of odd length?
Answer:	Before we can define a shortest closed walk of odd length, we need to know that the graph contains such closed walks in the first place: in an empty set of walks, there is no shortest walk!

To conclude this section, let's consider two examples of bipartite graphs, and how we prove that they are bipartite. In one case, we will use the definition directly; in the other, we will use Theorem 13.1.

Proposition 13.2. *For all $n \geq 1$, the hypercube graph Q_n is bipartite.*

Proof. The vertices of Q_n are bit strings $b_1 b_2 \dots b_n$ where each b_i is either 0 or 1. To define a bipartition (A, B) of $V(Q_n)$, we place a vertex $b_1 b_2 \dots b_n$ in A if $b_1 + b_2 + \cdots + b_n$ is even, and in B if $b_1 + b_2 + \cdots + b_n$ is odd.

Each edge of Q_n joins two vertices x and y that differ only in one position; $x_i \neq y_i$ for some i, but $x_j = y_j$ for all $j \neq i$. As a result,

$$(x_1 + x_2 + \cdots + x_n) - (y_1 + y_2 + \cdots + y_n) = x_i - y_i = \pm 1 :$$

every difference $x_j - y_j$ with $j \neq i$ cancels. Since the sums differ by ± 1, one of them is even and one is odd: the edge xy has one endpoint in A and one in B. Therefore (A, B) is a bipartition of Q_n, completing the proof. □

Question:	What is the relationship between this bipartition and distances in Q_n?
Answer:	This definition places a vertex in A whenever it is at an even distance from vertex $00 \dots 0$.

Proposition 13.3. *All trees are bipartite.*

Proof. We know from Theorem 10.2 that trees have no cycles at all—in particular, they have no odd cycles, so they are bipartite by Theorem 13.1. □

In Chapter 10, we already proved an equivalent statement to Proposition 13.3: it was Proposition 10.7. Our proof back then was much longer, because we did not have Theorem 13.1 at our disposal!

13.4 Matching problems

To discuss problems such as the domino puzzle and the rook puzzle, we make the following definition.

Definition 13.3. *A **matching** M in a graph G is a spanning subgraph of G in which every vertex has degree 0 or 1: no two edges in $E(M)$ share an endpoint.*

A vertex $x \in V(G)$ with $\deg_M(x) = 1$ is **covered** by M, and **uncovered** otherwise (when $\deg_M(x) = 0$). The vertices covered by M are exactly the endpoints of the edges in $E(M)$.

When it comes to a matching, we are almost exclusively interested in the edge set $E(M)$: it tells us where to place rooks or dominoes. It is common to simply define a matching to be a set of edges, no two sharing an endpoint. However, the spanning subgraph M is occasionally convenient to use; it is easy to go from M to $E(M)$, but not so easy to go from $E(M)$ back to M, so I have chosen the subgraph and not the set of edges as the fundamental object.

We can think about matchings in all kinds of graphs. But we will start by thinking about matchings in bipartite graphs for two reasons. First, many applications will naturally give us a bipartite graph in which to look for a matching. Second, the theory turns out to be simpler for bipartite graphs.

The empty subgraph M with no edges is also a matching by our definition—it's just not a very good one. It is more difficult, but also more interesting, to select more edges. The larger a matching is, the more vertices it covers, and so the absolute limit is to cover every vertex of G: this is what we're looking for in both examples we've seen so far.

Definition 13.4. *A **perfect matching** M in a graph G is a matching that covers every vertex of G.*

The bipartite matching problem can be considered from two points of view. The first is the optimistic point of view, in which we ask whether a perfect matching exists. This is very common in theoretical applications, including applications to other areas of math.

There is also what I call the "realistic" point of view. In many practical applications, while a perfect matching is still the best option, a very big matching is still good if it is not perfect. For example, one practical bipartite matching problem occurs in every math department when instructors are assigned to courses. It might not be possible in a given semester to offer every course, but does the math department give up and cancel classes for the semester? No! Instead, the department simply tries to schedule as many courses as possible.

Question:	What bipartite graph can we use to model assigning instructors to courses?
Answer:	The vertices of the graph are the course sections and the instructors that can teach them; the edges represent which courses an instructor is qualified to teach.

(Further complications on top of the matching problem arise when we consider giving an instructor multiple sections, which must not be scheduled at the same time.)

In general, instead of asking whether a perfect matching exists, the second point of view is to ask how big the largest matchings are. It's this point of view that we will start with in this textbook. Of course, a complete answer to this question will also tell us whether a perfect matching exists!

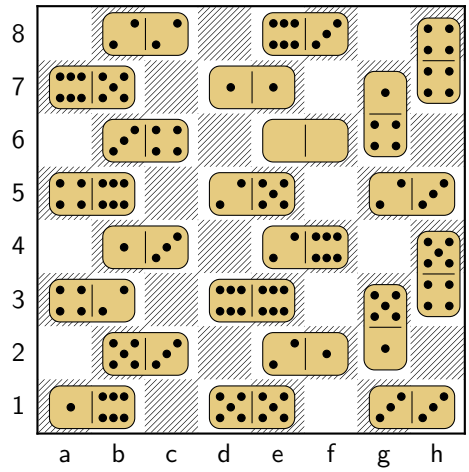


Figure 13.3: A clumsy packing of dominoes

13.5 Maximum and maximal

In combinatorial optimization problems like the matching problem, we are trying to find an object X that's as large as possible, subject to some condition. It's common to say that X is:

- *maximum* if it really is as large as you can get; there are no better solutions than X.
- *maximal* if you cannot add anything to X without violating the restriction; there might be better solutions, but you cannot get to them without removing something from X first.

Every maximum solution is also maximal, but the reverse might not hold.

In the same way, the words *minimum* and *minimal* get used when we're trying to pick a set that's as small as possible, subject to some condition.

In this textbook, I will also use this terminology, but I will avoid conveying important information solely by the difference between "maximum" and "maximal". Usually, we are interested in maximum and minimum objects, because we want to solve the optimization problems we pose, and I will let this situation pass by without further comment. If we really need a maximal object that is not necessarily maximum, or a minimal object that is not necessarily minimum, I will specifically point this out for you.

However, no matter which terminology you use, it's important to realize that there is a difference, and in particular there is a difference in the matching problem. For example, consider the domino placement in Figure 13.3. It is not a maximum matching in the 8×8 grid graph: with no squares removed, it is easy to get a perfect matching by, for example, placing 32 horizontal dominoes. However, it is a maximal matching: there is no empty space to add another domino.

In fact, in the case of dominoes, a much harder problem is the clumsy packing problem: what is the minimum number of edges in a maximal matching of the $n \times n$ grid graph? The diagram in Figure 13.3 shows a construction found by Gyárfás, Lehel, and Tuza in 1988 [45].

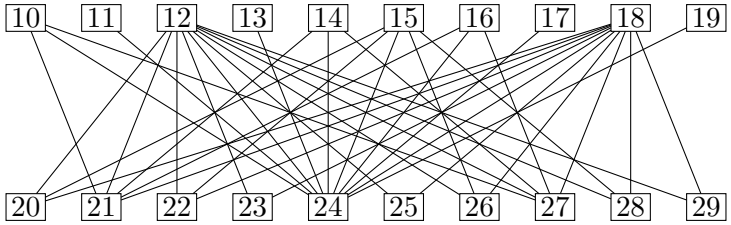


Figure 13.4: The multiples-of-six graph

Looking at this example also tells us that the bipartite matching problem cannot be solved by a greedy algorithm that selects the first edges it sees. We should expect to have to work hard before we find a general solution.

13.6 Vertex covers

Here is a new, more abstract example of a bipartite matching problem. Let G be the bipartite graph with vertices $\{10, 11, \ldots, 19\}$ on one side, $\{20, 21, \ldots, 29\}$ on the other side, and edges defined by the following rule: vertices x and y are adjacent when they begin with different digits and the product $x \cdot y$ is divisible by 6. A diagram of the multiples-of-six graph is shown in Figure 13.4, though it is a little busy in places. What is the largest matching in this graph?

Question:	Is there a perfect matching in this graph?
Answer:	No: vertices 11 and 13 on the top are only adjacent to vertex 24, so they cannot both be covered by a matching.

Question:	Look for a matching greedily: for each of the vertices $10, 11, \ldots, 19$, match it to the first unused vertex on the other side, if you can. What matching do you get?
Answer:	The matching whose edges are $\{10, 21\}$, $\{11, 24\}$, $\{12, 20\}$, $\{14, 27\}$, $\{15, 22\}$, and $\{18, 23\}$.

We've made a good start, but we have a gap: we've found a matching with 6 edges, but we can only prove that the matching can't have more than 9 edges. (That is, we've ruled out a perfect matching with 10 edges.) We either need to find a better solution, or we need to prove a better upper bound. Is there a bottleneck—some kind of limited resource that we exhaust if we try to find a larger matching?

Yes! The limited resource is the vertices divisible by 3. To get a product divisible by 6, we need both a multiple of 2 and a multiple of 3, but the multiples of 3 are much harder to come by: they are $12, 15, 18, 21, 24$, and 27. Every edge in a matching needs to use one of these vertices, so there can be at most 6 edges in a matching.

This argument generalizes.

Definition 13.5. *A **vertex cover** in a graph G is a set of vertices $U \subseteq V(G)$ such that every edge of G has one or both endpoints in U.*

For example, in the multiples-of-six graph, the set $\{12, 15, 18, 21, 24, 27\}$ is a vertex cover. It is not the only one. For example, the set of all vertices in a graph is guaranteed to be a vertex cover. The hard task is not to find a vertex cover, but to find a vertex cover that is as small as possible.

Vertex covers are important because, just as in the example of the multiples-of-six graph, we can use them to prove upper bounds on the size of a matching:

Proposition 13.4. *If M is a matching and U is a vertex cover in the same graph G, then $|E(M)| \leq |U|$.*

Proof. Consider the sum

$$\sum_{x \in U} \deg_M(u).$$

Each term $\deg_M(x)$ of this sum is at most 1, by the definition of a matching, so the value of the sum is at most $|U|$: the number of vertices in U.

However, $\deg_M(u)$ counts the number of edges of M incident to x, so we can also think of the sum as counting the number of times any edge of M is incident to any vertex of U. Each edge of M must be incident to at least one vertex of U, by the definition of a vertex cover, so each edge of M contributes at least 1 to the sum. Therefore the sum is at least $|E(M)|$ the number of edges in M.

Putting these inequalities together, we conclude that $|E(M)| \leq |U|$. □

Question:	Does Proposition 13.4 only apply to bipartite graphs?
Answer:	No: it is true for any graph.

In a bipartite graph G, however, there are two particularly easy examples: if (A, B) is a bipartition of G, then A and B are both vertex covers. By Proposition 13.4, there cannot be a matching with more than $\min\{|A|, |B|\}$ edges; in particular, there can only be a perfect matching if $|A| = |B|$.

Question:	What does this tell us about the domino problem on an 8×8 chessboard with two opposite squares removed?
Answer:	A complete 8×8 chessboard has 32 squares of each color, but two opposite squares have the same color; if they are removed, only 30 squares of that color are left. So at most 30 dominoes can be placed: not enough to cover the remaining 62 squares.

Question:	Can you use vertex covers to show that the 4×5 rectangle in Figure 13.1b cannot be avoided?
Answer:	Take $U = \{5, 6, 7, 8, f, g, h\}$. These four rows and three columns cover the entire chessboard except for the highlighted rectangle, showing that at most 7 rooks can be placed outside the rectangle.

Proposition 13.4 can be used with any matching M and any vertex cover U. However, if we want to get upper bounds on the size of a matching, then the smaller U is, the better the bounds we get. To get the best upper bound, we should try to find the smallest vertex cover possible. However, even for the smallest vertex cover in a graph, this bound might not tell us the full truth about matchings.

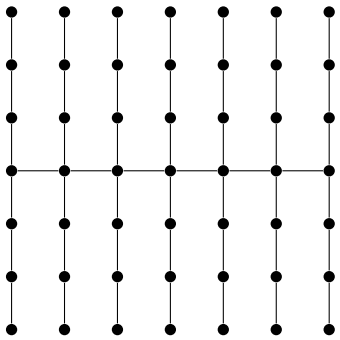
Question:	In the complete graph K_{100}, how many edges are in the largest matching?
Answer:	There are 50, because a 50-edge matching covers all 100 vertices: it is perfect.

Question:	In the complete graph K_{100}, how many vertices are in the smallest vertex cover?
Answer:	There are 99. We can leave out any one vertex from the vertex cover, but a set U missing two vertices x and y does not contain either endpoint of the edge xy.

It will turn out that in bipartite graphs, the problems of finding the largest matching and the smallest vertex cover do "meet in the middle". This is a result known as Kőnig's theorem (Theorem 14.2), which we will prove in the next chapter. Kőnig's theorem is the reason we are studying matchings in bipartite graphs first.

13.7 Practice problems

1. In the graph shown below, find a matching M and a vertex cover U with $|M| = |U|$.



2. Mathematicians that study arrangements of non-attacking rooks on chessboards of various shapes often define an object called the rook polynomial. This is a polynomial $p(x)$ in which the coefficient of x^k is the number of ways to place k non-attacking rooks onto the chessboard.

   a) Find the rook polynomial of a 5×5 chessboard.

   b) Find the rook polynomial of a 5×5 chessboard with one square missing.

3. The "cop-and-highway-robber game"[22] is played on a bipartite graph. One player (the highway robber) chooses an edge of the graph and occupies it for highway robbery. Simultaneously and without seeing the edge chosen, the other player (the cop) chooses a vertex of that edge to guard. If the cop guards an endpoint of the edge chosen by the highway robber, the highway robber gets caught and loses; otherwise, the highway robber gets away and wins.

   This is a game, like rock-paper-scissors, where both players need to choose their strategy at random; if either player plays predictably, then the other player can exploit this.

   a) Suppose that the graph has a matching M. Find a strategy for the highway robber to win with probability $1 - \frac{1}{|M|}$, no matter what the cop does.

   b) Suppose that the graph has a vertex cover U. Find a strategy for the cop to win with probability $\frac{1}{|U|}$, no matter what the highway robber does.

   (Why is this another proof of Proposition 13.4?)

4. a) Prove that when n is a multiple of 3, it is possible to place $\frac{1}{3}n^2$ dominoes on an $n \times n$ chessboard so that there is no more room to place another domino.

   b) Prove that whenever dominoes are placed on an $n \times n$ chessboard so that there is no more room to place another domino, at least $\frac{1}{3}(n^2 - n)$ dominoes must have been used.

5. Prove that a bipartite graph with minimum degree at least 2 contains a cycle of length at least 4.

6. Let G be a bipartite graph with bipartition (A, B). Prove the "bipartite handshake lemma":
$$\sum_{x \in A} \deg(x) = \sum_{x \in B} \deg(x) = |E(G)|.$$

7. Coins are placed on several of the squares on an 8×8 chessboard (at most one coin per square). Every row of the chessboard has the same number of coins on it, but every column of the chessboard has a different number of coins (possibly 0). How many coins are on the chessboard?

8. To explore the difference between maximum and maximal matchings, consider the following graph (which can be made arbitrarily large):

[22] Whose name I made up, inspired by the study of cops-and-robbers games on graphs, which are something fancier.

a) Find a perfect matching in this graph.

b) Find the smallest matching in this graph above which is maximal (there is no larger matching that contains all of its edges).

c) Let G be a graph (not necessarily bipartite) and let M be a maximal matching in G. Prove that G has a vertex cover U with $|U| = 2|M|$.

d) Prove that the example in this problem is essentially as bad as it gets: if M is a maximal matching, then there is no matching with more than $2|M|$ edges.

9. (BMO 2005) The equilateral triangle ABC has sides of integer length n. The triangle is completely divided (by drawing lines parallel to the sides of the triangle) into equilateral triangular cells of side length 1.

   A continuous route is chosen, starting inside the cell with vertex A and always crossing from one cell to another through an edge shared by the two cells. No cell is visited more than once. Find, with proof, the greatest number of cells which can be visited.

14 Kőnig's theorem

The purpose of this chapter

This chapter presents the algorithmic view of matchings in bipartite graphs. The augmenting path algorithm is the most serious algorithm seen in this textbook so far; accordingly, I've given it the most serious treatment, with a detailed proof that the algorithm works correctly, and an example of how the algorithm is used. I do not intend to treat every algorithm similarly; this is not a computer science textbook. This is, however, an introductory graph theory textbook, so I hope that this chapter introduces you to the algorithmic side of graph theory.

In my opinion, Kőnig's theorem (Theorem 14.2) is the right setting to consider the algorithmic questions, because it is a result comparing two optimization problems, which is how we usually encounter matchings in algorithmic applications. Of course, Kőnig's theorem has its uses in pure mathematics, and Hall's theorem (Theorem 15.1 in the next chapter) can also be proved by giving an algorithm. However, most uses of Hall's theorem are about abstractly proving the existence of a perfect matching in a general, incompletely specified family of graphs—and that is the sort of problem we will consider in the next chapter.

Kőnig's theorem will also become important in Chapters 25 through 28. We will use it to prove Menger's theorem (Theorem 27.1). In the other direction, algorithms for the maximum flow problem can be used to solve the bipartite matching problem.

14.1 Kőnig's theorem

As we've already seen in Chapter 4 with the notation for minimum degree ($\delta(G)$) and maximum degree ($\Delta(G)$), graph theorists like to use Greek letters for numerical invariants of graphs. We have recently learned two new such invariants, so let's learn the notation for them.

Definition 14.1. *The* ***matching number*** $\alpha'(G)$ *of a graph G is the number of edges in a maximum (largest) matching of G.*

Definition 14.2. *The* ***vertex cover number*** $\beta(G)$ *of a graph G is the number of vertices in a minimum (smallest) vertex cover of G.*

Unfortunately, the letters α ("alpha") and β ("beta") here have no particular meaning to help you remember them. There is a meaning to the apostrophe present in $\alpha'(G)$ but not in $\beta(G)$: it indicates that the invariant is an "edge version" of another invariant. This means that you can already begin to guess what $\alpha(G)$ (the "vertex version" of the matching number) might mean: later on, in Chapter 18, you will learn if your guess was right.

Using the new notation, we can rewrite Proposition 13.4, which we stated as an inequality between $|E(M)|$ (the number of edges in a matching M) and $|U|$ (the number of vertices in a vertex cover U). The new version reads:

Proposition 14.1. *In any graph G, $\alpha'(G) \leq \beta(G)$.*

Actually, it might not be immediately obvious that Proposition 13.4 and Proposition 14.1 are the same: the first is talking about an arbitrary matching and vertex cover, and the second is only talking about maximum matchings and minimum vertex covers.

However, if Proposition 13.4 holds for all M and U, then in particular it holds when M is a maximum matching and U is a minimum vertex cover, which proves Proposition 14.1. Conversely, if Proposition 14.1 holds, then for any matching M and vertex cover U, we have $|E(M)| \leq \alpha'(G) \leq \beta(G) \leq |U|$, proving Proposition 13.4. (The inequalities $|E(M)| \leq \alpha'(G)$ and $\beta(G) \leq |U|$ come from the meanings of the words "maximum" and "minimum".)

The main theorem of this chapter was proved in 1931 by Dénes Kőnig [62]. Actually, it is ambiguous which result you mean if you say "Kőnig's theorem", because Kőnig proved many results in graph theory: he was, in fact, the author of the first graph theory textbook [63], written in 1936.[23] In this textbook, the theorem we'll call Kőnig's theorem is the following:

Theorem 14.2 (Kőnig's theorem)**.** *For any bipartite graph G, $\alpha'(G) = \beta(G)$.*

We will prove this theorem by finding an algorithm. Given a matching M, the algorithm will either construct a larger matching M', or find a vertex cover U with $|E(M)| = |U|$.

Question:	If we find such a U, what can we conclude?
Answer:	Since every matching has at most $\|U\|$ edges, and M has exactly $\|U\|$ edges, we know that M is a maximum matching by Proposition 13.4. By similar reasoning, U is a minimum vertex cover.

However, we know that we cannot always improve a sub-optimal matching simply by adding edges. So what does it take to improve a sub-optimal matching?

14.2 Augmenting paths

We will need a set-theoretic operation that is not as well-known as union and intersection, but is the natural operation to consider when tracking changes. (This remains true whether we're tracking changes made to a matching in a graph, or tracking changes made to a Wikipedia entry, for example.) At its most basic, it's defined on sets: if A and B are two sets, then their **symmetric difference $A \oplus B$** is the set $(A \cup B) - (A \cap B)$: the set containing all elements in A, or in B, but not in both A and B.

[23] By a happy coincidence, this is 200 years after Euler first modeled a problem—the problem of the seven bridges of Königsberg—using what we'd now call a graph.

IMO 2024	IMO 2025	2024 ⊕ 2025
United States of America	People's Republic of China	Belarus
People's Republic of China	United States of America	United Kingdom
Republic of Korea	Republic of Korea	Hungary
India	Japan	Japan
Belarus	Poland	Israel
Singapore	Israel	Vietnam
United Kingdom	India	
Hungary	Singapore	
Poland	Vietnam	
Türkiye	Türkiye	

Figure 14.1: Top ten IMO teams in 2024 and in 2025

We can think of $A \oplus B$ as a summary of the changes that need to be made to A to get B. If we start with A, and "toggle" all the elements of $A \oplus B$ (removing them from A if they are in A, and adding them to A if not), then the result is B.

Question: If we know A and $A \oplus B$, what operation on them lets us find B?

Answer: It's the symmetric difference again: B is $A \oplus (A \oplus B)$. The elements in $A \oplus B$ but not A are the elements only in B, which we need to add to A; the elements in both A and $A \oplus B$ are exactly the ones not in B, which we need to remove.

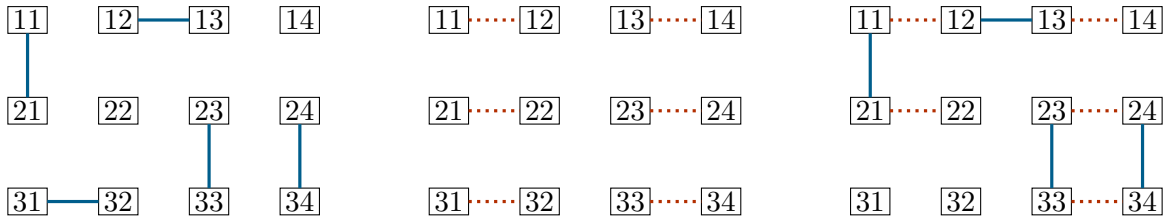
Question: Which elements does a triple symmetric difference $(A \oplus B) \oplus C$ contain?

Answer: The elements contained in an odd number (one, or all three) of the sets A, B, C. This rule generalizes, and can also be used to prove that $\oplus$ is associative: $(A \oplus B) \oplus C = A \oplus (B \oplus C)$, because the rule applies to both sides.

For example, Figure 14.1 shows a table listing the top ten countries in the International Mathematical Olympiad in 2024 and in 2025. If we think of these lists as sets, we can take the symmetric difference, shown in the third column. This symmetric difference shows the six countries whose status changed from 2024 to 2025: Belarus, the United Kingdom, and Hungary left the top ten list, while Japan, Israel, and Vietnam entered it.

Rather than continue thinking about competition performance, let's switch to thinking about graphs. Given two graphs G and H, the **symmetric difference $G \oplus H$** is the graph with vertex set $V(G \oplus H) = V(G) \cup V(H)$ and edge set $E(G \oplus H) = E(G) \oplus E(H)$. We're taking the symmetric difference of the edge sets, which are the star of the show; we take the union of the vertex sets to be sure that we've included the endpoints of every edge we need.

We want to use this idea to see what changes need to be made to get from one matching to another. We are looking for small, incremental changes. So we will compare two matchings M and N such that M only has one more edge than N.



(a) Matching M (b) Matching N (c) $M \oplus N$

Figure 14.2: Two matchings in a 3×4 grid

Figure 14.2 shows one such example. To make the drawing more clear, I have not included the edges of the original graph (which you may assume to be a 3×4 grid graph). Matching M (in Figure 14.2a) has only 5 edges, while N (in Figure 14.2b) is a perfect matching with 6 edges.

As a graph, $M \oplus N$ (shown in Figure 14.2c) consists of four connected components. Two of them are just isolated vertices, and don't have a lot to tell us. Another, with the vertices $\{23, 24, 33, 34\}$, is a cycle, and represents only that the matchings M and N match those four vertices in two different ways.

The last component is the most interesting one: the path represented by $(22, 21, 11, 12, 13, 14)$. It's this component that tells us how to improve matching M: we must swap the two edges $\{11, 21\}$ and $\{12, 13\}$ for the three edges $\{11, 12\}$, $\{13, 14\}$, and $\{21, 22\}$.

We will call such a subgraph an M-augmenting path (or just an augmenting path, if it is clear what matching M it is meant to augment). In general, if M is a matching in a graph G, then an ***M*-augmenting path** is a path in G with the following properties:

- The path begins and ends at two vertices that are not covered by M.
- The edges used by the path alternate between edges not in M and edges in M.

In a symmetric difference of two matchings, we can also see connected components (paths or cycles) that satisfy only the second property, but not the first: we call such paths and cycles *M-alternating*.

The length of an M-augmenting path must be odd, or else it will not end at an uncovered vertex. If it has length $2k + 1$, then the $1^{\text{st}}, 3^{\text{rd}}, \ldots, (2k + 1)^{\text{th}}$ edges are not part of M, while the $2^{\text{nd}}, 4^{\text{th}}, \ldots, (2k)^{\text{th}}$ edges are.

If we find a path P in $M \oplus N$ that helps us improve M to be more like N, then we should apply that difference to M by taking the symmetric difference $M \oplus P$. This is generalized by the following lemma:

Lemma 14.3. *Let M be a matching in a graph G and let P be an M-augmenting path. Then $M \oplus P$ is a matching with one more edge than M.*

Proof. We verify that $M \oplus P$ is a matching by checking the degree of every vertex in $M \oplus P$.

Let P be an $x - y$ path; then $\deg_P(x) = \deg_P(y) = 1$. Because P is an M-augmenting path, $\deg_M(x) = 0 = \deg_M(y) = 0$. Therefore x and y have degree 1 in $M \oplus P$ as well; they each get an incident edge from P, and do not gain or lose anything from M.

For every vertex $z \in V(P)$ other than x and y, the path P has two edges incident to z, one of which is in M and the other is not (because P is M-alternating). In $M \oplus P$, the edge in M is lost, but the edge not in M is kept, so $\deg_{M \oplus P}(z) = 1$.

Finally, all vertices not on the path are unaffected by the change from M to $M \oplus P$: they still have the same degree they had in M, which is at most 1. Therefore $M \oplus P$ is a matching.

Looking above, we can see that vertices x and y have degree 1 in $M \oplus P$, but degree 0 in M; for every other vertex, its degree in M and $M \oplus P$ is the same. Therefore $M \oplus P$ is a matching that covers two more vertices than M, which means it has one more edge than M. □

In general, given two matchings M and N, the symmetric difference $M \oplus N$ is a graph with maximum degree 2: each vertex has at most one incident edge in each matching. The connected components of such a graph are M-alternating paths and M-alternating cycles.

Suppose now that, as in our example, N is a slight improvement over M: $|E(N)| = |E(M)| + 1$. Then, after canceling out the edges common between M and N, the symmetric difference $M \oplus N$ still has one more edge from N than from M. So there must also be at least one connected component of $M \oplus N$ where this is true!

Question:	If a connected component of $M \oplus N$ is a cycle, can it contain more edges from N than from M?
Answer:	No: to be M-alternating, exactly half of its edges must be from M, and exactly half must be from N.

Question:	If a connected component of $M \oplus N$ is a path, can it contain more edges from N than from M?
Answer:	Only if it is an M-alternating path of odd length where the first and last edge both come from N. This is an M-augmenting path!

In particular, this argument proves that there exists an M-augmenting path, but not usefully: to find it, we needed to know the bigger matching N. In our proof of Kőnig's theorem, we will see how to find an M-augmenting path in a more helpful fashion, but only in bipartite graphs.

14.3 The augmenting path algorithm

Let G be a bipartite graph with bipartition (A, B), and let M be a matching in G. The main tool we need to prove Kőnig's theorem is an algorithm for finding an M-augmenting path.

Our algorithm will be yet another breadth-first search algorithm. However, we will not explore the graph along all possible trajectories, but only along ones that can form an M-augmenting path.

We are searching for a path between two uncovered vertices, so let's introduce some notation to help us. Write A as $A_0 \cup A_1$, where the vertices in A_0 are uncovered by M and the vertices

in A_1 are covered by M. (That is, $x \in A_k$ if x has degree k in M.) Split B into B_0 and B_1 in the same way.

Since an augmenting path has odd length, and every path in G alternates between A and B, an augmenting path must have one end in A_0 and one end in B_0. We will arbitrarily choose to start our exploration in A_0, with the goal of eventually reaching a vertex in B_0.

To keep track of our progress, let A_{exp} and B_{exp} be the set of **exp**lored vertices in A and in B, respectively. These will change over the course of the algorithm. Since we start our exploration in A_0, initially $A_{\text{exp}} = A_0$ and $B_{\text{exp}} = \varnothing$.

Finally, we will also want to know how we reached the explored vertices: when we add a new vertex x to $A_{\text{exp}} \cup B_{\text{exp}}$, we make note of $f(x)$, the vertex **f**rom which we explored x. This will help us find an augmenting path, if one exists.

The augmenting path algorithm repeats the following steps, which I've given short names to help us refer to them.

1. **Explore:** For each vertex $a \in A_{\text{exp}}$, explore all its neighbors: if a neighbor b of a is not already in B_{exp}, add it, and set $f(b) = a$. (After the first iteration, this only needs to be done for the vertices newly added to B_{exp}.)
2. **Check:** At this step, there are two stopping conditions to check, which conclude the algorithm with different results.
   - **Path?** If there is a vertex $b \in B_{\text{exp}} \cap B_0$ (an explored vertex uncovered by M), stop and return the path represented by the walk
     $$(b, f(b), f(f(b)), \dots)$$
     that traces back the vertices from which we reached b, ending at a vertex in A_0. We will show that this is an M-augmenting path.
   - **Cover?** If no new vertices were added to B_{exp} in the most recent **Explore** step, stop and return the set U defined[24] to be $(A - A_{\text{exp}}) \cup B_{\text{exp}}$. We will show that U is a vertex cover and $|U| = |E(M)|$.
3. **Match:** Otherwise, $B_{\text{exp}} \subseteq B_1$: all the vertices in B_{exp} are covered by M. For each vertex b added to B_{exp} in the most recent **Explore** step, find the edge $ab \in E(M)$ incident to b. Explore vertex a (the vertex that b is matched to): add it to A_{exp}, and set $f(a) = b$.

To prove the correctness of this algorithm, we must show that when the **Path** or **Cover** stopping conditions are met, the path really is an M-augmenting path and the cover really is a vertex cover of the same size as M.

Question:	We must also show that the algorithm cannot loop forever. Why must it halt?
Answer:	At every iteration in which we don't stop by the **Cover** condition, we add a new vertex to B_{exp}, and $\lvert B_{\text{exp}}\rvert$ is limited in size by $\lvert B\rvert$.

[24] When learning about this algorithm for the first time, this definition may feel like it came out of nowhere! In the next section, we'll discuss where this formula for the vertex cover comes from.

In practice, this algorithm is applied multiple times. Starting from an initial matching, which does not need to be very good, we repeatedly use the algorithm to grow the matching by one edge. Eventually, we either find a perfect matching, or find a vertex cover that proves we cannot proceed any further. This self-certifying feature, in addition to being theoretically useful in the proof of Kőnig's theorem, is also practically convenient: it means that we can verify the output of the algorithm independently if we're not certain that an implementation of it is bug-free.

Which initial matching do we start with? One option is to begin with the empty matching: $E(M) = \varnothing$.

Question:	What does the algorithm do if $E(M) = \varnothing$?
Answer:	We start with all vertices of A already in A_{exp}, and all vertices of B in B_0. If even a single edge exists, we will explore a vertex of B_0 and halt in the first step, returning a path of length 1.

Question:	When a path of length 1 is an M-augmenting path, what does this mean?
Answer:	It means that the edge on that path can be added to M without removing any edges.

Starting from the empty matching, not just the first time we augment the matching but the first few times will typically end equally quickly: they will find an edge between two uncovered vertices. In effect, this is no different from a greedy algorithm. Only once the greedy algorithm would give up, because it cannot add any more edges, does the augmenting path algorithm start finding longer augmenting paths.

14.4 Analyzing the algorithm

A common technique when analyzing an algorithm is to prove an invariant of the algorithm. Just like a graph invariant is a property that does not change when the graph is relabeled, an invariant of an algorithm is also a property that does not change—it does not change over the course of the algorithm. Appendix A discusses this proof technique, but this is the first time we really need it in the main body of the textbook.

There are two invariants we track for the augmenting path algorithm: one invariant to help us find an M-augmenting path, and one invariant that

For the first invariant, we define a path $P(x)$ for each vertex $x \in A_{\text{exp}} \cup B_{\text{exp}}$. It is the path represented by

$$(x, f(x), f(f(x)), \dots)$$

that starts at x and traces back the vertices from which we reached x, ending at a vertex in A_0. Recall that in the **Path** stopping condition, this is the path we return, for a vertex $x \in B_{\text{exp}} \cap B_0$.

Lemma 14.4. *For any vertex $x \in A_{\text{exp}} \cup B_{\text{exp}}$, the path $P(x)$ is an M-alternating path.*

Proof. More precisely, we prove the following: each edge $yf(y)$ is in $E(M)$ if $y \in A_{\text{exp}}$, and not in $E(M)$ if $y \in B_{\text{exp}}$. The path $P(x)$ must alternate between A_{exp} and B_{exp}: like any other path in G, it alternates between A and B, and it only passes through explored vertices. So this claim will prove that $P(x)$ is an M-alternating path.

Let $a \in A_{\text{exp}}$ be a vertex on $P(x)$. Then either $a \in A_0$ (in which case, it is the end of the path) or else a was explored from a vertex $f(a) \in B_{\text{exp}}$. We explore from vertices in B_{exp} in a **Match** step; therefore $af(a) \in E(M)$.

Let $b \in B_{\text{exp}}$ be a vertex on $P(x)$. Then b was explored from a vertex $f(b) \in A_{\text{exp}}$; we must prove that $bf(b) \notin E(M)$. If $f(b) \in A_0$, this is automatic, because then M has no edges incident to $f(b)$. Otherwise, $f(b)$ was itself explored from some vertex $f(f(b))$, and as we've proved above, $f(b)f(f(b)) \in E(M)$. Therefore $bf(b) \notin E(M)$, because $f(b)$ cannot be incident to two edges of M. This completes the proof. □

Next, we turn our attention to the set $U = (A - A_{\text{exp}}) \cup B_{\text{exp}}$, which we return if the **Cover** stopping condition is satisfied. First of all, let's understand where this set comes from.

Question:	If a vertex cover U satisfies $\lvert U\rvert = \lvert E(M)\rvert$, can U ever use both endpoints of an edge in M?
Answer:	No: if we select just one endpoint of each edge in M, we've already selected $\lvert E(M)\rvert$ different vertices. Choosing both endpoints of an edge in M is wasteful and we cannot afford it.

Question:	If a vertex cover U satisfies $\lvert U\rvert = \lvert E(M)\rvert$, how must it treat A_0 and B_0?
Answer:	U cannot contain any vertices in A_0 and B_0. Again, we've already reached $\lvert E(M)\rvert$ vertices just by selecting a vertex from each edge in M. We can't select any more without going over our limit, so we can't select any vertices not covered by M.

These two guiding questions tell us what to do to the vertices we explore. Our initial set A_{exp} consists just of A_0: these vertices cannot be part of U. This means that to cover the edges with an endpoint in A_0, we must add all their other endpoints to U. These neighbors are exactly what we put in B_{exp} in the **Explore** step. Next, if a vertex b we've added to U is an endpoint of an edge $ab \in E(M)$, we can't put the other endpoint of ab in U as well, so $a \notin U$. So the vertices matched to B_{exp} cannot be part of U, and these are exactly the vertices we put in A_{exp} in the **Match** step.

This logic continues: at each step, vertices we put in A_{exp} are vertices we know cannot be part of U, and vertices we put in B_{exp} are vertices we know must be part of U. Since we don't know anything about vertices outside $A_{\text{exp}} \cup B_{\text{exp}}$ a reasonable first try is to define $U = (A - A_{\text{exp}}) \cup B_{\text{exp}}$, and we we will see that this turns out to work.

We can define $U = (A - A_{\text{exp}}) \cup B_{\text{exp}}$ at any point over the course of the algorithm, not just at the end. It is not a vertex cover before the end, but it satisfies an invariant of its own:

Lemma 14.5. *Immediately before every* ***Explore*** *step,* $|U| = |E(M)|$.

Proof. At the beginning of the algorithm, $A_{\text{exp}} = A_0$ and $B_{\text{exp}} = \varnothing$; therefore $U = A - A_0 = A_1$. Every edge of M has one endpoint in A_1 and one endpoint in B_1; therefore $|U| = |A_1| = |E(M)|$.

In the **Explore** step, some vertices are added to B_{exp}. Then, in the **Match** step, for every vertex b that we added to B_{exp}, we add a vertex a to A_{exp}: the vertex a such that $ab \in E(M)$. Such a vertex a cannot already have been in A_{exp}: it is not in A_0 (because $ab \in E(M)$), and so it can only be explored in the **Match** step, and only from b.

Therefore, if we add k vertices to B_{exp} in the **Explore** step, we also add k vertices to A_{exp} in the **Match** step (provided we do not stop before then). As a result, $|A - A_{\text{exp}}|$ decreases by k and $|B_{\text{exp}}|$ increases by k, so $|U| = |A - A_{\text{exp}}| + |B_{\text{exp}}|$ remains unchanged

Since $|U|$ starts equal to $|E(M)|$, and a single **Explore**–**Check**–**Match** iteration of the algorithm does not change $|U|$, we must also have $|U| = |E(M)|$ before every **Explore** step. □

Many uses of invariants to study algorithms are proofs by induction in disguise, and you can see that in the proof of Lemma 14.5. Here, we begin with a base case: we show that $|U| = |E(M)|$ at the beginning of the algorithm. Next, we show that whenever this property holds, it still holds after one more iteration.

Together, Lemma 14.4 and Lemma 14.5 are not enough to draw any conclusions about what happens at the end of the algorithm: we need to see how the stopping conditions contribute.

Proposition 14.6. *If the* ***Path*** *stopping condition holds, then for a vertex* $b \in B_{\text{exp}} \cap B_0$*, the path* $P(b)$ *is an* M*-augmenting path.*

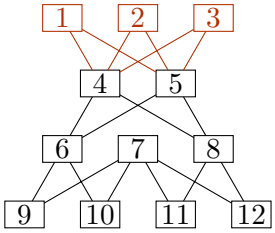
Proof. We already know from Lemma 14.4 that $P(b)$ is an M-alternating path. Moreover, its endpoints are uncovered by M: its start is b, which is in B_0, and its end can only be a vertex in A_0, because all other explored vertices were explored from somewhere. Therefore $P(b)$ is an M-augmenting path. □

Proposition 14.7. *If the* ***Cover*** *stopping condition holds, then* $U = (A - A_{\text{exp}}) \cup (B_{\text{exp}})$ *is a vertex cover with* $|E(M)| = |U|$.

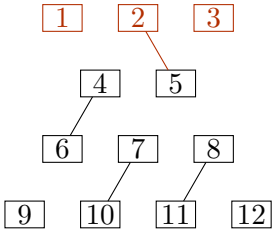
Proof. To show that U is a vertex cover, take any edge $ab \in E(G)$, where $a \in A$ and $b \in B$. If $a \in A_{\text{exp}}$, then in the most recent explore step, we would have made sure that $b \in B_{\text{exp}}$ (if it was not already there for some other reason); in this case, $b \in U$. But if $a \notin A_{\text{exp}}$, then $a \in U$. In either case, one of the endpoints of ab is in U.

From Lemma 14.5, we know that $|E(M)| = |U|$ before any **Explore** step. If the **Cover** stopping condition holds, then no new vertices were added to A_{exp} in the explore step, so $|U|$ did not change; therefore $|E(M)| = |U|$ at the end of the algorithm. □

This completes the proof that the algorithm is correct. It also gives us all the tools we need to prove Kőnig's theorem.



(a) The volcano graph



(b) A 4-edge matching M

Figure 14.3: The volcano graph and an initial matching

Proof of Theorem 14.2. Let M be a maximum matching in the bipartite graph G; starting with the matching M, run the augmenting path algorithm.

The algorithm cannot find an M-augmenting path, because by Lemma 14.3, we could use it to obtain a bigger matching, which contradicts our choice of M. So it must stop by satisfying the **Cover** stopping condition. By Proposition 14.7, the algorithm outputs a vertex cover U with $|E(M)| = |U|$.

Since $\alpha'(G) = |E(M)|$ (by our choice of M) and $\beta(G) \leq |U|$ (the minimum vertex cover is at least as small as U), we conclude that $\alpha'(G) \geq \beta(G)$. From Proposition 14.1, we know $\alpha'(G) \leq \beta(G)$; therefore the two are equal. □

14.5 Using the algorithm

Algorithms like the augmenting path algorithm are not really intended to be performed by hand. Very rarely, if a graph is neither so small that we can find a matching without any algorithms, nor too big to deal with by hand, we can use the algorithm to check a matching we suspect to be optimal. Most of the time, though, you should use a computer. So the purpose of the example here is simply as an illustration, so that you can better understand the algorithm and its proof by seeing it in action. (You will gain even better understanding if you try doing this yourself, and I've included a practice problem about this. After you understand the algorithm perfectly, there's no point.)

The graph we will use as an example is a graph I've called *volcano graph*, shown in Figure 14.3a. It does not really have any special properties, but I defined it and named it and I have a hat with the volcano graph on it, so it is my favorite. One of my whims about the definition is that I insist that the top of the volcano graph must always be drawn in red.

Question:	Why is the volcano graph bipartite?
Answer:	All edges have one endpoint in $\{1, 2, 3, 6, 7, 8\}$ and one endpoint in $\{4, 5, 9, 10, 11, 12\}$.

I have mentioned that the first few rounds of finding augmenting paths are essentially equivalent to a greedy algorithm, so I will skip them to avoid wasting time on cases that don't help

us understand anything. We will begin with the matching M shown in Figure 14.3b, with $E(M) = \{\{2,5\},\{4,6\},\{7,10\},\{8,11\}\}$.

We must make an arbitrary choice between the two sides: let's say that side A (the side that we explore from) is $\{1,2,3,6,7,8\}$. For our initial matching, we start with $A_0 = \{1,3\}$ and $B_0 = \{9,12\}$.

The algorithm proceeds as follows:

1. **Explore** vertices 4 and 5, adding them to B_{exp}, with $f(4) = 1$ and $f(5) = 3$.
2. **Check** the stopping conditions:
   - Is there a **Path?** No: $B_{\text{exp}} \cap B_0 = \varnothing$.
   - Is there a **Cover?** No: we added 2 new vertices to B_{exp}.
3. **Match** vertex 4 to vertex 6 and vertex 5 to vertex 2. Add 2 and 6 to A_{exp}, with $f(2) = 5$ and $f(6) = 4$.
4. **Explore** vertices 9 and 10, with $f(9) = f(10) = 6$. As before, add them to B_{exp}.
5. **Check** the stopping conditions:
   - Is there a **Path?** Yes: $B_{\text{exp}} \cap B_0 = \{9\}$.

   We stop, and output the augmenting path represented by

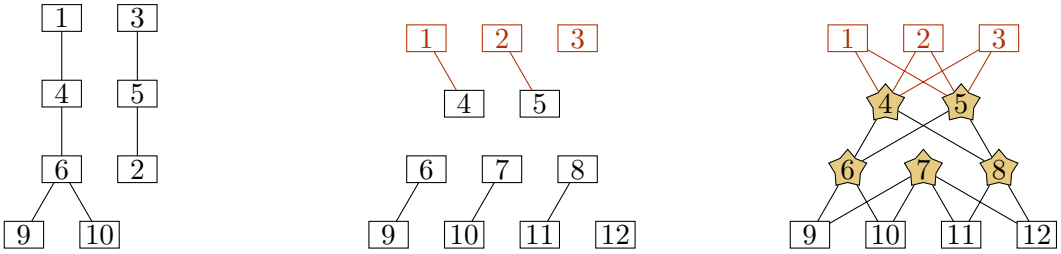
$$(9, f(9), f(f(9)), f(f(f(9)))) = (9,6,4,1).$$

Of the edges on the augmenting path, edge $\{4,6\}$ is part of M, while edges $\{1,4\}$ and $\{6,9\}$ are not. Therefore we improve our matching M by removing $\{4,6\}$, adding $\{1,4\}$ and $\{6,9\}$ instead. The resulting matching (call it M') is shown in Figure 14.4b.

It is common to represent the exploration process by a forest, with the vertices arranged in levels. At the top level are the vertices in A_0; from there, the levels alternate between vertices added to B_{exp} in an **Explore** step and vertices added to A_{exp} in a **Match** step. The forest we get in this example is shown in Figure 14.4a.

Question:	How can we tell which edges of this forest are edges of M?
Answer:	They are the edges whose top vertex is in a B_{exp} level and whose bottom vertex is in an A_{exp} level. In Figure 14.4a, these are just the edges down from $\{4,5\}$, but if we had kept going, any edges down from $\{9,10\}$ would also have been in M.

Question:	The forest in Figure 14.4a looks like it has other augmenting paths, such as the one represented by $(10,6,4,1)$. Could we have used this path instead?
Answer:	No: this path only looks like an augmenting path because we stopped before the next **Match** step, but in reality, vertex 10 is covered by M. If we had kept going, we would have found a longer augmenting path.



(a) The exploration forest (b) A 5-edge matching M' (c) The vertex cover U

Figure 14.4: An example of the augmenting path algorithm

We have improved the matching, but we're still not done, so we continue with another round of the algorithm. The edges of the new matching are $\{\{1,4\},\{2,5\},\{6,9\},\{7,10\},\{8,11\}\}$, so the uncovered vertices are given by $A_0 = \{3\}$ and $B_0 = \{12\}$. Now we:

1. **Explore** vertices 4 and 5, adding them to B_{exp}, with $f(4) = f(5) = 3$.
2. **Check** the stopping conditions:
   - Is there a **Path?** No: $B_{\text{exp}} \cap B_0 = \varnothing$.
   - Is there a **Cover?** No: we added 2 new vertices to B_{exp}.
3. **Match** vertex 4 to vertex 1 and vertex 5 to vertex 2. Add 1 and 2 to A_{exp}, with $f(1) = 4$ and $f(2) = 5$.
4. **Explore**, but fail to find any vertices: vertices 1 and 2 are only adjacent to vertices 4 and 5, which are already in B_{exp}.
5. **Check** the stopping conditions:
   - Is there a **Path?** No: $B_{\text{exp}} \cap B_0 = \varnothing$.
   - Is there a **Cover?** Yes: we did not add any new vertices to B_{exp}.

   We stop, and output the vertex cover

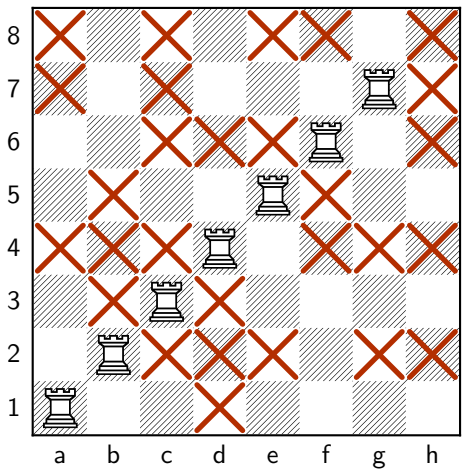
$$U = (A - A_{\text{exp}}) \cup B_{\text{exp}} = \{6,7,8\} \cup \{4,5\}.$$

Indeed, the vertex $U = \{4,5,6,7,8\}$ shown in Figure 14.4c really is a vertex cover: it consists of the middle two layers of the volcano graph. Every edge between the first two layers is incident to 4 or 5, every edge between the last two layers is incident to 6, 7, or 8, and every edge between the second and third layer has both endpoints in U. Since $|U| = |E(M')| = 5$, we conclude that M' is a maximum matching.

14.6 Practice problems

1. Use the augmenting path algorithm to improve the matching in Figure 14.2a. Does this result in the perfect matching in Figure 14.2b, or a different one?
2. Viewing the domino tiling in Figure 13.3 as a matching, find a set of 10 augmenting paths (with no vertices in common) that will transform it into a perfect matching.

3. On the chessboard below, how many rooks can be placed so that no two rooks occupy the same row or column, and all the crossed-out squares are empty?



A 7-rook solution is shown. Use the augmenting path algorithm to find either an 8-rook solution, or a vertex cover proving that the 7-rook solution is optimal.

4. Let G be a bipartite graph with bipartition (A, B) and minimum degree $\delta(G)$; let U be a vertex cover that does not contain either A or B.

   a) Prove that $|U| \geq 2\delta(G)$. When does it follow that $\alpha'(G) \geq 2\delta(G)$, and why?

   b) In the case $|A| = |B| = 10$ and $\delta(G) = 3$, give an example of a bipartite graph in which this lower bound is true: $\alpha'(G) = 6$.

5. If G is not bipartite, then Proposition 14.1 still guarantees that $\alpha'(G) \leq \beta(G)$, but the two quantities may be different.

   a) Find $\alpha'(G)$ and $\beta(G)$ when $G = C_{2k+1}$, a cycle graph with an odd number of vertices, in terms of k. Verify that they are, in fact, different.

   b) Find a graph G which satisfies $\alpha'(G) = \beta(G)$, but is not bipartite.

6. Let G be a bipartite graph G with bipartition (A, B) which is r-regular: every vertex has degree r. Let $n = |A|$.

   a) Prove that $|B| = n$ as well, and find $|E(G)|$ in terms of n and r.

   b) Prove that a set of k vertices in G can cover at most kr of the edges. What does this say about the size of a vertex cover?

   c) Use Kőnig's theorem to prove that G must have a perfect matching.

7. Prove if the augmenting path algorithm is applied to a maximum matching, then A_{exp} is the set of all vertices on side A which are left uncovered by some maximum matching in the graph.

   (You must prove two things: that if $a \in A_{\text{exp}}$, then there is some maximum matching which does not cover a, and that if $a \in A - A_{\text{exp}}$, then all maximum matchings cover a.)

15 Hall's theorem

The purpose of this chapter

Applications of bipartite matchings to other areas of math are often all-or-nothing: either we find a perfect matching, or else we have made no progress. Additionally, the graphs we face are not arbitrary: they obey some mathematical laws defining their edges.

In such a scenario, Hall's theorem is usually the way to proceed. Hall's condition is a necessary and sufficient condition for the matching we want, and the abstract definition of our graph means it's often easy to verify. In this chapter, we will deduce Hall's theorem from Kőnig's theorem and then—because this is not enough to make a complete chapter—we will see some applications, ranging from recreational mathematics to serious combinatorics.

Hall's theorem is also often referred to as Hall's marriage theorem, and Hall's condition is then called the marriage condition. I'm not philosophically opposed to this (it can sometimes be handled poorly when teaching, but it can also be handled well), but I don't like the terminology, so I've left it out of this chapter.

15.1 Hall's theorem

Chapter 14 was devoted to proving Kőnig's theorem (Theorem 14.2), which is the result you want to use for finding a maximum matching in a bipartite graph. In many applications, that is absolutely the wrong question to ask, because only a perfect matching counts as a solution; if no perfect matching exists, we don't care if the largest matching leaves 2 vertices or 992 vertices uncovered.

Of course, Kőnig's theorem can also tell us when a perfect matching exists, in terms of the number of vertices in a minimum vertex cover, but this feels like the wrong way of thinking. In a graph G with bipartition (A, B), where $|A| = |B| = n$, we want to distinguish between $\beta(G) = n$ and $\beta(G) \leq n - 1$ to determine whether a perfect matching exists, and this doesn't feel like a dramatic change. What's more, a vertex cover containing $n - 1$ vertices isn't a very intuitive explanation of why there is no perfect matching.

However, there are some situations in which we can tell a clear story about why a perfect matching fails to exist.

Question:	What if the graph contains an isolated vertex, x?
Answer:	Then there is no perfect matching, because there is no matching that covers x.

Question:	What if vertices x and y have degree 1 and share the same neighbor z?
Answer:	Then a matching can contain edge xz and cover x, or contain edge yz and cover y, but not both: so there is still no perfect matching.

Generalizing these concepts, we can describe a minimum requirement for a graph to have a perfect matching. First, we need a new definition.

Definition 15.1. *The **neighborhood** $N_G(S)$ of a set $S \subseteq V(G)$ in a graph G is the set of all vertices of G adjacent to at least one vertex in S. If the graph G is clear from context, we simply write $N(S)$.*

For example, a vertex x is isolated if $N(\{x\}) = \varnothing$. Two leaf vertices x and y share their only neighbor z if $N(\{x, y\}) = \{z\}$. What do these situations have in common? A set of vertices has a neighborhood which is too small: $N(S)$ is smaller than S itself.

Question:	Let S be an arbitrary subset of A. Can there be a perfect matching if $\lvert N(S)\rvert < \lvert S\rvert$?
Answer:	No: a perfect matching must pair each vertex of S with a different vertex in $N(S)$, which is impossible if there are not $\lvert S\rvert$ different vertices in $N(S)$ to use.

This explains why this condition is necessary for a perfect matching to exist. Hall's theorem (proved by Philip Hall in 1935 [47]) states that the condition is also sufficient:

Theorem 15.1 (Hall's theorem)**.** *A bipartite graph G with the bipartition (A, B) has a matching that covers all vertices in A if and only if it satisfies Hall's condition: for all subsets $S \subseteq A$, $\lvert N(S)\rvert \geq \lvert S\rvert$.*

In particular, when $\lvert A\rvert = \lvert B\rvert$, G has a perfect matching if and only if it satisfies Hall's condition.

Question:	What if $\lvert A\rvert > \lvert B\rvert$?
Answer:	Then Hall's condition will not be satisfied if we take $S = A$, because $N(A) \subseteq B$ and therefore $\lvert N(A)\rvert < \lvert A\rvert$.

Question:	What if $\lvert A\rvert < \lvert B\rvert$?
Answer:	Then there is no perfect matching. In this case, Hall's theorem promises us a matching that covers every vertex in A, but leaves some vertices in B uncovered.

Applying Hall's theorem when $|A| < |B|$ can still be very useful if the vertices in A are different in type from the vertices in B. For example, in the magic trick, we really want a matching that covers every vertex $\{a, b, c\}$ corresponding to a set of 3 cards: then the matching tells my assistant what to do when handed such a set. It would be fine if the matching did not cover some ordered pairs (a, b): that would mean that my assistant will never hand me those two cards.

By proving Kőnig's theorem, we have done all the hard work of proving Hall's theorem, and can now deduce it as a corollary. (It would also have been possible to go the other way, and prove Kőnig's theorem as a corollary of Hall's theorem.)

Proof of Theorem 15.1. We already know that Hall's condition is necessary for a matching to cover A to exist: if Hall's condition is violated for some subset $S \subseteq A$, then there is not even a matching that covers every vertex in S. It remains to prove that Hall's condition is sufficient. We will do this by assuming that there is no matching that covers A, and finding a set S for which Hall's condition is violated.

Every edge in a matching has one endpoint in A and one endpoint in B. So if there is no matching that covers A, then there is no matching with $|A|$ edges: $\alpha'(G) < |A|$. By Kőnig's theorem, $\beta(G) < |A|$: there is a vertex cover U such that $|U| < |A|$.

This set U is a small set of vertices with a lot of neighbors, since every edge of G has at least one endpoint in U. To find a set S for which Hall's condition is violated, we want the opposite: a large set of vertices with few neighbors. So we define S to be $A - U$: the set of all vertices in A that are not part of the vertex cover.

For all vertices $x \in S$, if y is adjacent to x, then U contains one endpoint of xy, but U does not contain x, so U must contain y. Therefore all of $N(S)$ is contained in U. In addition to the vertices of $N(S)$ (which are all on side B), U contains all of $A - S$ (on side A); therefore

$$|U| \geq |N(S)| + |A - S| = |N(S)| + |A| - |S|.$$

However, we also know $|U| < |A|$, so $|N(S)| + |A| - |S| < |A|$. This can be rearranged to get $|N(S)| < |S|$, proving that S violates Hall's condition. □

Question:	Suppose I give you a set S which violates Hall's condition. Can you use it to find a vertex cover with fewer than $\|A\|$ vertices?
Answer:	Yes: the set $(A - S) \cup N(S)$ is a vertex cover. For every edge xy, where $x \in A$ and $y \in B$, we have two cases: if $x \in S$, then y is in the vertex cover, and if $x \notin S$, then x is in the vertex cover.

15.2 Regular bipartite graphs

It is important to be clear about how Hall's theorem is meant to be used. It is not meant to be wielded like a hammer, going through the subsets of A one by one and searching for one that violates Hall's condition. Such an algorithm would take an exponentially long time! No: if you have a concrete graph in front of you, and it has no short mathematical description, use the

augmenting path algorithm in Chapter 14 to find a maximum matching and a minimum vertex cover. (There's one exception: if you can quickly spot a small set that violates Hall's condition, of course you should use that.)

Instead, we use Hall's theorem when dealing with a graph or a family of graphs in which the edges follow a regular mathematical pattern, and we can prove that Hall's condition is satisfied.

Our first example predates both Kőnig's theorem and Hall's theorem: it originally appeared in 1894, in a thesis by Ernst Steinitz about point-line configurations [94], though it was not stated in the language of graph theory.

Theorem 15.2. *For all $r \geq 1$, if G is an r-regular bipartite graph, then it has a perfect matching.*

I admit that I am so fond of this theorem that I've put two versions of it in a practice problem already, once in Chapter 8 and once in Chapter 14. But we have not had the opportunity to use Hall's theorem to prove it, so let's do that.

Proof of Theorem 15.2. Here's a starting observation that proves it's permissible to at least hope for a perfect matching. If G has a bipartition (A, B), then we can count the edges of G in two ways. First, we could sum the degrees of all vertices in A and get $r|A|$: this counts every edge once, because every edge has an endpoint in A. We could also sum the degrees of all vertices in B and get $r|B|$. The two methods must give the same answer, so $r|A| = r|B|$, and $|A| = |B|$.

To prove that a perfect matching exists, we show that Hall's condition holds for an arbitrary subset $S \subseteq A$. As in the previous paragraph, there are $r|S|$ edges with an endpoint in S. Similarly, there are $r|N(S)|$ edges with an endpoint in $N(S)$. By definition, every edge with an endpoint in S has the other endpoint in $N(S)$; in other words, the $r|N(S)|$ edges with an endpoint in $|N(S)|$ include among them all $r|S|$ edges with an endpoint in S. This gives the inequality $r|N(S)| \geq r|S|$, or $|N(S)| \geq |S|$. □

Theorem 15.2 is one of the few results in matching theory that is useful for multigraphs, not just simple graphs. You would not expect this to happen! After all, loops and parallel edges can never help us find a larger matching: a matching has maximum degree 1, which means it is always a simple graph. In fact, bipartite graphs cannot contain loops in the first place, though they can have parallel edges: a loop can never have one endpoint in A and the other in B.

Question:	So why would it matter that Theorem 15.2 is true for multigraphs?
Answer:	It is possible that a bipartite multigraph is only r-regular due to the presence of parallel edges, and if we eliminate those, then checking Hall's condition becomes much harder.

I should emphasize that when we go from an r-regular multigraph to a simple graph, Hall's condition will still be true: it will just no longer follow automatically from Theorem 15.2.

To confirm for ourselves that Theorem 15.2 still does work for multigraphs, we could go through all our proofs and verify that nothing changes in the presence of parallel edges. (We would need to go through many proofs because of the dependencies between them.) However, there is a shortcut.

Let G be a bipartite r-regular multigraph, and let G' be the simplification of G. We can quickly check that the two-paragraph proof of Theorem 15.2 still works for multigraphs, and therefore G satisfies Hall's condition. Therefore G' also satisfies Hall's condition: adjacent vertices in G are still adjacent in G', so neighborhoods don't change at all. Therefore G' has a perfect matching, which corresponds to a perfect matching in G.

It is possible to generalize the result of Theorem 15.2. Suppose that some vertices in A can have degree more than r, and some vertices in B can have degree less than r. Then $r|S|$ is a lower bound on the number of edges with an endpoint in S, while $r|N(S)|$ is an upper bound on the number of edges with an endpoint in $N(S)$, but this only extends the inequalities we have; we are still able to obtain $r|N(S)| \geq r|S|$ and verify Hall's condition.

Question:	Does this give us a perfect matching?
Answer:	No, because it's now possible that $\|A\| < \|B\|$.

However, we can still obtain the following corollary:

Corollary 15.3. *For all r, if G is a bipartite graph with bipartition (A, B) such that every vertex in A has degree at least r, while every vertex in B has degree at most r, then G has a matching that covers A.*

Armed with Hall's theorem and several of its variants, we are now ready to tackle some problems outside graph theory.

15.3 A three-card magic trick

Here is a mathematical card trick I know. A version of it using the ordinary 52-card deck was invented by Fitch Cheney [68], but I will present a simplified version with a deck of 8 cards, numbered 1 through 8.

You choose 3 cards from the deck, however you like, and give them to my assistant, while I look away or even temporarily leave the room. The assistant then hands me two of the cards, one at a time, and hands the third card back to you, for reference.

I have not seen the third card you're holding, at any point. Nevertheless, I can now perform some quick mental arithmetic and name the number on that card!

How is this possible? The answer doesn't involve any hidden communication: my assistant doesn't try to communicate the third card via eyebrow gestures and split-second timing. The answer involves only mathematics. Before describing a particular implementation of this trick, let's understand how it is even possible to make the trick work.

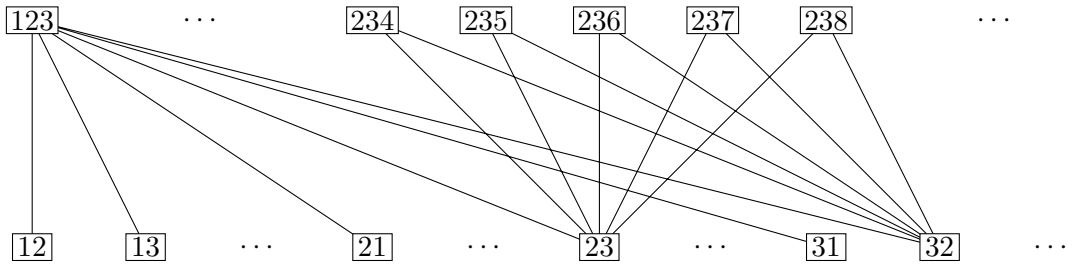


Figure 15.1: The bipartite graph for the three-card magic trick

Consider the following bipartite graph. On one side of the bipartition, the vertices are sets of three cards: $\{1, 2, 3\}$ through $\{6, 7, 8\}$. In the magic trick, you choose three cards, which we can represent by a choice of one of these vertices. On the other side, the vertices are ordered pairs of two cards: $(1, 2)$, $(1, 3)$, and so on through $(8, 7)$. In the magic trick, this represents the information I get: the order in which the assistant hands me the cards makes the pair an ordered pair.

For the edges, join each set of three cards to the 6 ordered pairs that can be formed from it: these represent the 6 choices my assistant has, in deciding which two cards to give me and in which order. The graph has 112 vertices, so it is too big to draw in full, but Figure 15.1 shows a portion of it: enough to see all the neighbors of vertices $\{1, 2, 3\}$ on the first side and of $(2, 3)$ and $(3, 2)$ on the second side.

My assistant's strategy, whatever it might be, picks a designated neighbor of each vertex on the first side of the graph; for example, if $(1, 2)$ is the designated neighbor of $\{1, 2, 3\}$, then if you choose the cards numbered 1, 2, and 3, my assistant will hand me card 1 and then card 2. Let M (for "magic") be the spanning subgraph containing only the edges between a vertex on the first side and its designated neighbor; by construction, every vertex on the first side has degree 1 in M.

I presumably know M ahead of time, so if I am handed card 1 and then card 2, I should think about the edges of M incident to vertex $(1, 2)$. If the edge between $\{1, 2, 3\}$ and $(1, 2)$ is the only such edge in M, then I know that the third card must have been 3. If, however, there are multiple edges of M incident to $(1, 2)$, then I have many theories, and cannot identify the third card. In other words, the magic trick only works if every vertex on the second side has degree at most 1 in M: if M is a matching!

Does such a matching exist? A first step is to check the number of vertices. There are $\binom{8}{3} = \frac{8 \cdot 7 \cdot 6}{3!} = 56$ vertices on the first side: the number of ways to choose a set of 3 objects out of 8. There are $8 \cdot 7 = 56$ vertices on the second side: the number of ways to choose a pair of 2 distinct objects out of 8. So it's conceivable to have a matching M that covers all 56 vertices on the first side, as long as it also covers all 56 vertices on the second side: it must be a perfect matching. We haven't ruled out the possibility that the magic trick works.

Question:	Why do we divide by 3! in $\frac{8\cdot 7\cdot 6}{3!}$, but don't divide by anything in $8 \cdot 7$?
Answer:	In the first case, we're counting sets, which are unordered, so we divide by the 3! ways to order the elements. In the second case, we're counting ordered pairs: my assistant hands me a first card and a second card.

To see that the graph for the 8-card magic trick has a perfect matching, all we have to do is observe that it is 6-regular and apply Theorem 15.2.

Question:	Why is the graph 6-regular?
Answer:	For every pair of cards I see, there are 6 possibilities for the missing third card. For every three cards my assistant sees, there are 6 possible actions: 3 ways to choose the first card to pass me, multiplied by 2 ways to choose the second card.

We can generalize to a k-card magic trick: you choose k cards from a (potentially very large) deck, my assistant hands me $k-1$ of them in some order, and I must guess the last card.

Proposition 15.4. *The k-card magic trick can be performed if the deck contains at most $k! + k - 1$ cards.*

Proof. Suppose the deck contains n cards; construct the bipartite graph with bipartition (A, B) where A is the set of all unordered k-card hands, B is the set of all ordered sequences of $k-1$ cards, and there is an edge whenever a k-card hand contains all cards in the $(k-1)$-card sequence.

Then every vertex in A has degree $k \cdot (k-1)! = k!$: when my assistant takes a k-card hand and hands me $k-1$ cards in some order, there are k ways to choose which card I don't get to see, and $(k-1)!$ ways to sort the cards I do see. Every vertex in B has degree $n-k+1$: when I am handed $k-1$ cards, the number of k-card hands they could have come from is equal to the number of possibilities for the k^{th} card, which is $n-(k-1)$ because that card can be any of the cards except for the $k-1$ I see.

If we let $r = k!$, then every vertex in A has degree at least (actually, exactly) r. Provided $n-k+1 \leq r$, every vertex in B has degree at most r; this inequality is equivalent to $n \leq k!+k-1$. When this happens, by Corollary 15.3, the graph has a matching M that covers all vertices in A.

At least in principle, if my assistant and I agree on a matching M, then we have a strategy for the k-card trick. When handed k cards, my finds that k-card hand as a vertex in M, and looks at the only neighbor of that vertex to see which $k-1$ cards to give me, and in which order. When I am handed that sequence of $k-1$ cards, I can do the reverse: find that sequence as a vertex in M, and look at the only neighbor of that vertex to see which k cards were chosen. (If $n < k! + k - 1$, not all sequence of $k-1$ cards are covered by M, but every sequence my assistant actually hands is guaranteed to be covered!) This tells me, in particular, which card is the hidden card. □

Question:	What's the problem with implementing this strategy?
Answer:	For an arbitrary matching, too much memorization is necessary! Essentially, my assistant and I have to memorize what to do in every possible situation.

In 2002, Michael Kleber [61] not only proved Proposition 15.4, but also described a strategy for the k-card trick that can be implemented in practice, and I will finish our discussion of magic tricks by describing it in the three-card case.

My assistant's job is to add up the numbers on the three cards, and find the remainder when the sum is divided by 3. The card my assistant hands back to you is the smallest of the three cards if the remainder is 0, the middle card if the remainder is 1, and the largest card if the remainder is 2. My assistant hands me the other two cards in increasing order if the missing card is one of the three smallest cards I won't see, and in decreasing order otherwise.

Question:	What will my assistant do if you choose the cards $1, 3, 6$?
Answer:	$1+3+6 = 10$, which leaves a remainder of 1 when divided by 3, so my assistant will hand you the middle card: 3. Of the cards I won't see, the three smallest ones are $2, 3, 4$, of which 3 is one, so my assistant hands me 1, then 6.

My job is to label the unseen cards, in order from largest to smallest, as <2, <1, <0, >2, >1, and >0. I then guess the one where the inequality sign ($<$ or $>$) matches the relationship between the first and second card I am given, and the number (0, 1, or 2) matches the remainder when the sum of my two cards is divided by 3.

Question:	What will I do if I am handed the cards 1 and 6 in that order?
Answer:	Since $1 < 6$ and $1 + 6$ leaves a remainder of 1 when divided by 3, I want the card labeled <1: the second-smallest unseen card. So I guess card 3.

Though this strategy is workable, and generalizes well, I wonder if there's a simpler procedure specifically for the three-card magic trick. Maybe you will find one!

15.4 Generalized tic-tac-toe

The ordinary game of tic-tac-toe is played on a 3×3 grid. Players take turns placing their mark to claim an empty space in the grid: a $\times$ for the first player and a $\bigcirc$ for the second player. A player wins by claiming all three spaces in a horizontal, vertical, or diagonal line; if this does not happen when the board is filled, then the game is a draw.

The ordinary game of tic-tac-toe is not very interesting once you've learned how to play: every game between two skilled players ends in a draw. However, there are many ways to generalize

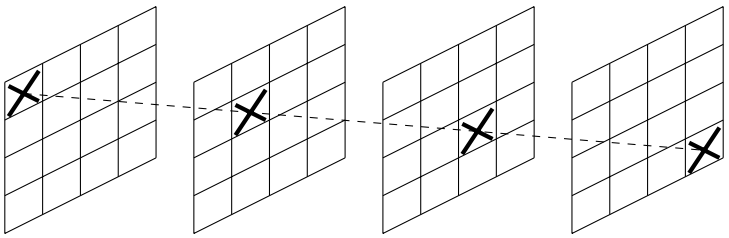


Figure 15.2: A winning line in $4 \times 4 \times 4$ tic-tac-toe

it. The game gomoku is played on a 15×15 board, and the winning condition is to get 5 consecutive pieces in a row. Mathematicians have also tried playing tic-tac-toe in three (or more?) dimensions. A $3 \times 3 \times 3$ game is not very interesting, because the first player has an insurmountable advantage when claiming the center space, but $4 \times 4 \times 4$ tic-tac-toe is worth playing. You just have to learn to spot the winning lines (one example is shown in Figure 15.2).

We will consider tic-tac-toe in even further abstraction: there is a set of points $\mathcal{P}$, which the players take turns claiming, and a set of winning lines $\mathcal{L}$. Each element of $\mathcal{L}$ is a subset of $\mathcal{P}$, and a player that claims all the elements of a winning line wins.

Graph theory cannot solve all possible tic-tac-toe games at once, but an application of Hall's theorem can let us stop thinking about games that are too boring: either player can easily guarantee a draw by a strategy with very little interaction.

Proposition 15.5. *In a generalized tic-tac-toe game where every set of k winning lines contain at least $2k$ points in total, either player can guarantee themselves at least a draw.*

Proof. We will construct a somewhat unusual bipartite graph to represent the game. On one side of the bipartition, we will have $\mathcal{P}$: the set of points. On the other side of the bipartition, we will not put $\mathcal{L}$, but rather two disjoint sets called $\mathcal{L}^+$ and $\mathcal{L}^-$. For every winning line $\ell \in \mathcal{L}$, we put a "positive vertex" ℓ^+ into $\mathcal{L}^+$ and a "negative vertex" ℓ^- into $\mathcal{L}^-$. Both ℓ^+ and ℓ^- will be adjacent to the same set of points in $\mathcal{P}$: all the points contained in ℓ.

Next, we set out to check Hall's condition for a matching in this graph to cover $\mathcal{L}^+ \cup \mathcal{L}^-$. Let $S \subseteq \mathcal{L}^+ \cup \mathcal{L}^-$ be an arbitrary set. Then either $|S \cap \mathcal{L}^+|$ or $|S \cap \mathcal{L}^-|$ is at least $\frac{1}{2}|S|$: either at least half the vertices in S are positive, or at least half the vertices are negative. The two situations are symmetric, so we assume $|S \cap \mathcal{L}^+| \geq \frac{1}{2}|S|$.

If $k = |S \cap \mathcal{L}^+|$, then there are k winning lines corresponding to vertices in $S \cap \mathcal{L}^+$, and by the condition we've assumed, there are at least $2k$ points on these lines. All these points are in $N(S \cap \mathcal{L}^+)$, and therefore in particular they are in $N(S)$. Therefore $|N(S)| \geq 2k \geq |S|$, and Hall's condition holds.

Hall's theorem gives us a matching in our unusually constructed graph, but what do we do with this matching? Well, for every line ℓ, the matching gives us two points on ℓ: the point matched to ℓ^+ and the point matched to ℓ^-. In other words, from every winning line we've selected two points, such that each point has been selected from at most one of the winning lines through it.

Using these selected points, you can obtain a draw whether you are the first player or the second. Suppose your opponent has played on one of the points selected from winning line ℓ. Then respond by claiming the other point selected from ℓ, if you have not claimed it already.

In all other cases—if your opponent's move is not the point selected from any line, or if you've already claimed the other point, or if it's the first move of the game—play arbitrarily. (A strategy of this type is called a pairing strategy in game theory.)

Question:	What if your opponent plays on that point with the goal of winning along some line other than ℓ?
Answer:	You don't care, because that other line has two selected points of its own.

If you use the pairing strategy, you will never lose, because your opponent can never claim all the points on any line. To do so, eventually your opponent would need to claim one of the selected points from that line, but then you will just claim the other point. You will probably not win, either, because the pairing strategy makes absolutely no effort to do so; it's possible that you will win accidentally. However, you are guaranteed at least a draw. □

The condition of Proposition 15.5 seems hard to check, but for many tic-tac-toe games, it turns out k winning lines will contain far more than $2k$ in most cases, leaving only a few cases to check. For example, consider a 5×5 board. A single line is guaranteed to have 5 points; a second line is guaranteed to add 4 new ones; a third line is guaranteed to add 3 new ones, for a total of 12.

Question:	How can we argue rigorously that 3 lines contain at least 12 points?
Answer:	They have $5 + 5 + 5 = 15$ points if we double-count the intersections, and 3 lines have at most 3 intersection points.

Since 12 points is enough for up to 6 lines, we can skip ahead to considering a set of 7 lines. These cover most of the board, and with a little bit more thinking, we can conclude that the worst case is a set of all $k = 12$ lines, which cover all 25 points. Therefore Proposition 15.5 applies to tic-tac-toe on a 5×5 board.

15.5 Incomparable sets

Our setting in this section will be the world of subsets of an n-element sets; we might as well assume they are subsets of $\{1, 2, \ldots, n\}$, because the exact elements won't matter.

Two subsets X and Y are called *comparable* if $X \subseteq Y$ or $Y \subseteq X$, and *incomparable* otherwise. For example, the set $\{2, 3\}$ is comparable to (and a subset of) the set $\{2, 3, 4\}$; it is incomparable to $\{1, 3, 4\}$. You can imagine that if the elements represent objects you want to own, then $\{2, 3\}$ is clearly not as good as $\{2, 3, 4\}$, but it is not certain which of $\{2, 3\}$ or $\{1, 3, 4\}$ is better. For example, what if elements 1, 3, and 4 are different kinds of cookies, while element 2 is an expensive car?

Suppose we want to pick a family[25] of sets in which no two sets are comparable. An example is the family $\big\{\{1,2,3\},\{2,4\},\{1,3,4\}\big\}$.

Question:	What is largest family of subsets of $\{1,2,3,4\}$ in which no two are comparable?
Answer:	$\big\{\{1,2\},\{1,3\},\{1,4\},\{2,3\},\{2,4\},\{3,4\}\big\}$, the family of all 2-element subsets.

In general, if we take the family of all k-element subsets, for any k, then any two subsets in the family will be incomparable. The number of sets in this family is given by the binomial coefficient $\binom{n}{k} = \frac{n!}{k!(n-k)!}$. This is maximized when $k = n/2$: for example, when $n = 4$ we have

$$\binom{4}{0} = 1,\ \binom{4}{1} = 4,\ \binom{4}{2} = 6,\ \binom{4}{3} = 4,\ \binom{4}{4} = 1.$$

So we want the family of all $(n/2)$-element subsets. We will call this the *middle layer* family.

Question:	What if n is odd?
Answer:	When n is odd, there are two equally good middle layers: the family of $\frac{n-1}{2}$-element subsets, and the family of $\frac{n+1}{2}$-element subsets. (For example, if $n = 5$, then $\binom{5}{2} = \binom{5}{3} = 10$.

By convention, let's round $n/2$ down if n is odd. The notation for rounding down is $\lfloor n/2 \rfloor$, so the formula $\binom{n}{\lfloor n/2 \rfloor}$ tells us the size of the middle layer family.

In 1928, Emanuel Sperner proved [93] that this is the best we can do, for any n. He did not use Hall's theorem to do this, but we will.

Theorem 15.6. *If $\mathcal{F}$ is a family of subsets of $\{1,2,\ldots,n\}$ such that no two different sets $X, Y \in \mathcal{F}$ are comparable, then $|\mathcal{F}| \leq \binom{n}{\lfloor n/2 \rfloor}$.*

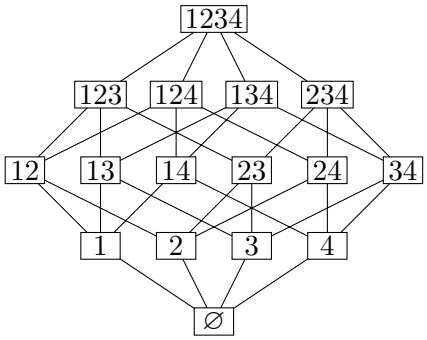
Proof. In Chapter 4, I mentioned that the hypercube graph Q_n can be useful when reasoning about set families, and that's what we will do here. We defined the vertex set of Q_n to be the set of n-bit strings: sequences $b_1 b_2 \ldots b_n$ where each b_i is either 0 or 1. To relate this to set families, a vertex $b_1 b_2 \ldots b_n$ can be identified with the subset of $\{1,2,\ldots,n\}$ containing an element i whenever $b_i = 1$. The edges, which join n-bit strings that differ in only one position, now tell us which two sets differ only by the presence or absence of one element. For the rest of the proof, I will switch to using set-related terminology. Figure 15.3a shows, for example, Q_4 with the vertices interpreted as subsets (and $\{x,y,z\}$ written as xyz to simplify the diagram).

We proved in Proposition 13.2 that Q_n as a whole is bipartite, but a perfect matching in Q_n will not help us. Instead, we will write Q_n as a union of bipartite graphs:

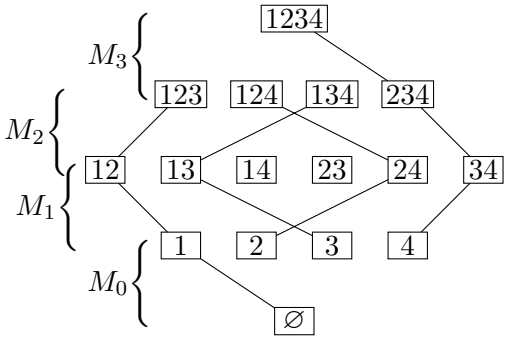
$$Q_n = G_{n,0} \cup G_{n,1} \cup \cdots \cup G_{n,n-1}$$

where $G_{n,k}$ is the subgraph of Q_n induced by the sets of size k or $k+1$. In other words:

[25] A "family" of sets is just a set of sets, but we give it a different word to make it easier to distinguish it from its elements.



(a) Q_4, interpreting vertices as subsets



(b) $H = M_0 \cup M_1 \cup M_2 \cup M_3$

Figure 15.3: Diagrams accompanying the proof of Theorem 15.6

- $G_{n,k}$ has the bipartition (A, B) where A is the set of k-element subsets of $\{1, 2, \ldots, n\}$, and B is the set of $(k+1)$-element subsets;
- An edge xy with $x \in A$ and $y \in B$ exists whenever we can add one more element to x to make y. (It will be important later that when this happens, x is a subset of y.)

In $G_{n,k}$, every vertex $x \in A$ has $n-k$ neighbors in B: there are $n-k$ elements of $\{1, 2, \ldots, n\}$ not already in x which we can add. Every vertex $y \in B$ has $k+1$ neighbors in A: it has $k+1$ elements which we can remove. Provided $n-k \geq k+1$, or $k \leq \frac{n-1}{2}$, Corollary 15.3 applies, giving us a matching in $G_{n,k}$ that covers A.

Question:	What kind of matching can we get if $k \geq \frac{n-1}{2}$?
Answer:	In this case, Corollary 15.3 would apply if we switched the roles of A and B, so there is a matching that covers B.

In either case, call this matching M_k. As a general rule, M_k covers the side of $G_{n,k}$ with fewer elements, which is the side further from the middle layer(s).

Now the magic happens! Consider the union $M_0 \cup M_1 \cup M_2 \cup \cdots \cup M_{n-1}$, and call it H. Figure 15.3b shows one possible union H in the case $n = 4$.

Consider a vertex of Q_n corresponding to a subset of size k, for any k. This vertex has degree 1 or 2 in H: if $k \leq \frac{n-1}{2}$, it is incident to one edge in M_k and maybe also one edge in M_{k-1}, while if $k \geq \frac{n-1}{2}$, then it is guaranteed one edge in M_{k-1} and possibly also an edge in M_k. We can follow these edges "up" the graph (increasing the number of elements) and "down" the graph (decreasing the number of elements) until we hit a dead end—a vertex of degree 1. So the connected component containing the vertex we chose is a path, which means that every component of H is a path.

How many connected components are there? Well, from each vertex, we can always follow edges of H to get closer to the middle layer(s): those are the guaranteed edges. Therefore each component of H contains one vertex in the middle layer (or in each middle layer, for odd n). There are $\binom{n}{\lfloor n/2 \rfloor}$ vertices in the middle layer, and therefore there are $\binom{n}{\lfloor n/2 \rfloor}$ components.

So far, we have only looked at the structure of subsets of $\{1, 2, \ldots, n\}$, but now we are ready to consider the set family $\mathcal{F}$ that Theorem 15.6 is all about. You see, $\mathcal{F}$ can never contain two

sets in the same connected component of H. If it did, then we could start at the set with fewer elements, and follow it up the path to get to the one with more elements. At each step, we're adding an element, so the first set will be a subset of the second—but we've assumed that $\mathcal{F}$ does not contain two such sets.

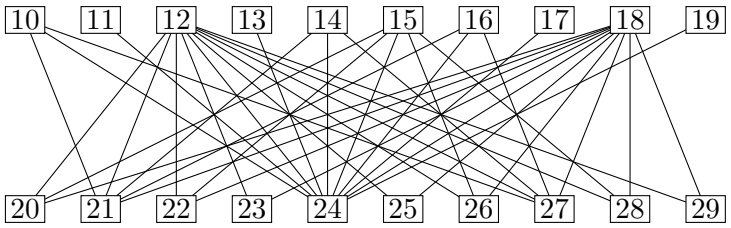
Therefore $\mathcal{F}$ contains at most one set from each connected component of H; since there are $\binom{n}{\lfloor n/2 \rfloor}$ components, we know that $\mathcal{F}$ contains at most $\binom{n}{\lfloor n/2 \rfloor}$ elements. □

15.6 Practice problems

1. Is it possible to circle a letter in each of the words below so that all 9 circled letters are different? Either find a way to do it, or prove that it's impossible using Hall's condition.

AREA	APART	ERRATA
GRAPH	PAPER	RETREAT
THEORY	TREE	YOGA

2. The multiples-of-six graph from Chapter 13 is shown again below:



   Prove that it does not have a perfect matching, but this time, using Hall's theorem.

3. Let G be a bipartite graph with n vertices on each side of the bipartition whose minimum degree $\delta(G)$ is greater than $n/2$.

   Prove that G has a perfect matching.

4. Prove that when $n > k! + k - 1$, it is impossible to perform the k-card magic trick with an n-card deck and guarantee that I guess the k^{th} card. (This is not a problem about graph theory; it is a matter of counting.)

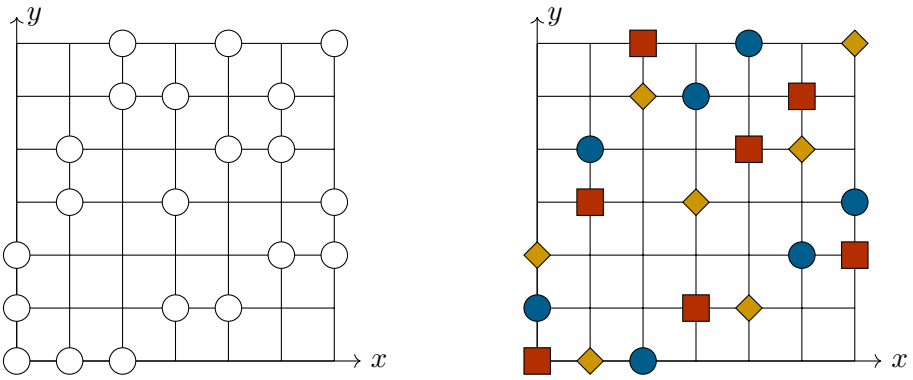
5. Find a pairing strategy for tic-tac-toe on a 5×5 board, where the goal is to win by claiming all 5 spaces along a horizontal, vertical, or diagonal line.

6. Let G be a bipartite graph, with bipartition (A, B), that has the following properties:

   - Every vertex on side A has degree 3 or 5;
   - Every vertex on side B has degree 2 or 4;
   - There are no edges between vertices of degree 3 and vertices of degree 4.

   Prove that G has a matching that covers all vertices in A.

7. Suppose that we mark several points at integer coordinates in the xy-plane in such a way that for every marked point (a, b), the lines $x = a$ and $y = b$ each contain two other marked points. This can be done by making a 3×3 grid, or in other, more complicated ways, such as in the first diagram below.



Prove that it is guaranteed to be possible to give the marked points 3 different colors, as in the second diagram above, so that no horizontal or vertical line passes through two marked points of the same color.

8. (Putnam 2012) A round-robin tournament of $2n$ teams lasted for $2n - 1$ days, as follows. On each day, every team played one game against another team, with one team winning and one team losing in each of the n games. Over the course of the tournament, each team played every other team exactly once. Can one necessarily choose one winning team from each day without choosing any team more than once?

9. An $n \times n$ Latin square is an $n \times n$ grid filled with the integers 1 through n in such a way that every row and every column contains each integer exactly once. For example, the first 5×5 grid below is a Latin square.

1	2	3	4	5
3	1	4	5	2
5	3	1	2	4
2	4	5	1	3
4	5	2	3	1

1	2	3	4	5
5	3	4	1	2

Suppose that, as in the second grid above, the first r rows in the $n \times n$ grid are filled with integers 1 through n in a way that does not cause any contradictions: each of the r rows use each integer exactly once, and no column contains any duplicates. Prove that we can fill in the last $n - r$ rows to get a Latin square.

16 Matchings in general graphs

The purpose of this chapter

In the previous three chapters, we discussed matchings in bipartite graphs; here, we are going to take another chapter to consider matchings in general graphs. This division is not because the generalization is easy, but because it is hard. We will have to do quite a bit of work just to prove Tutte's theorem (Theorem 16.1): the equivalent of Hall's theorem in cases where the graph is not necessarily bipartite.

There is a lot more that we're going to leave undone. We proved Kőnig's theorem which deals with the size of a maximum matching; we will not prove the Tutte–Berge formula, which generalizes Tutte's theorem to a Kőnig-type result. We used an algorithm to find maximum matchings in bipartite graphs. There is a more general algorithm, known as the blossom algorithm, which can handle non-bipartite graphs, but we will not discuss it; it is considerably more complicated. If you would like to learn more about matchings, I recommend Lovász and Plummer's book *Matching Theory* [70].

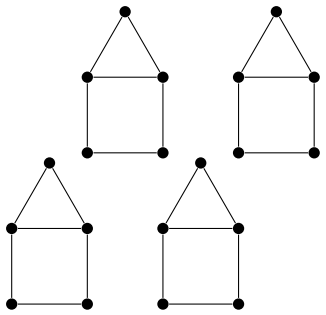
Compared to the material on Tutte's theorem, the second half of the chapter on 1-factorizations is less intense, and gives some graph-theoretical solutions to problems that come up in real life. A more relaxed path through this chapter is to learn about Tutte sets and the statement of Tutte's theorem, then dive into 1-factorizations and the geometric proof of Theorem 16.3.

There are a lot of other problems about factorizations I could have included instead of the section on increasing walks. In the end, though, I felt that many people could write a chapter about decomposing K_n into triangles or Hamilton cycles, but if I did not mention this beautiful problem in my textbook, it is not likely that someone else would write about it in theirs.

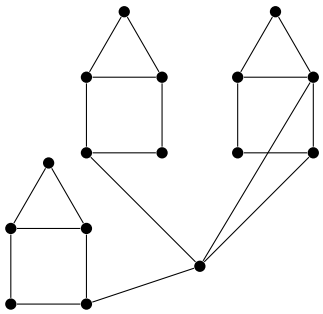
16.1 Tutte sets

In the case of bipartite graphs, we have a complete list of all the reasons that a perfect matching might fail to exist: the violations of Hall's condition. We can summarize these violations as problems of insufficiency: some vertices do not have enough neighbors for us to get a perfect matching.

If our graphs don't have to be bipartite, a second reason that we might not have a perfect matching appears, and that is parity. The complete graph K_{99}, or indeed K_{2n+1} for any n, does not have a perfect matching, even though it has all the edges we could wish for, simply because the number of vertices is odd. Each edge in a matching covers 2 vertices, and so the number of vertices covered by a matching is always even.



(a) Four houses



(b) Three houses and one more vertex

Figure 16.1: Obstacles to the existence of a perfect matching

Question:	Why didn't we need to worry about parity when proving theorems about bipartite graphs?
Answer:	In a graph with bipartition (A, B), we already know that a perfect matching only exists if $\|A\| = \|B\|$. When this happens, the total number of vertices is guaranteed to be even.

If we only had to worry about whether the total number of vertices is odd or even, that wouldn't be so bad. But there are examples showing that our life can be even worse:

Question:	In Figure 16.1a, the total number of vertices is even, and yet there's still a parity issue stopping us from having a perfect matching. How?
Answer:	Each house is a connected component with an odd number of vertices, so it has no perfect matching, and different houses cannot help each other.

We will use this idea throughout this chapter, and so we will say that a connected component with an odd number of vertices is an *odd component*, for short. However, odd components all by themselves are not the only problem.

Question:	In Figure 16.1b, there is a parity issue even though the graph has no odd components. How?
Answer:	Each of the three houses has no perfect matching on its own, and the extra vertex can be matched to only one of the houses.

We can imagine that the three houses in Figure 16.1b are on fire, and the vertex in place of the fourth house is a superhero that can put the fires out. However, the superhero can only be in one place, and cannot put out all the fires. This problem is a combination of parity and insufficiency: we have three parity problems in the graph, but only one vertex that can help us fix them.

We can think of Hall's theorem (Theorem 15.1) as saying that if a bipartite graph does not have a perfect matching, then we can summarize why, by giving an example where Hall's condition

fails. Similarly, we would like to have somewhere to point the finger of blame when a general graph does not have a perfect matching.

In both examples in Figure 16.1, we were able to tell a story about what went wrong, but can we generalize? The generalization was first found in 1947 by William Thomas Tutte [100].

We say that a set of vertices U in a graph G is a *Tutte set* if the graph $G-U$ has more than $|U|$ odd components.

Question:	Is there a Tutte set in Figure 16.1b?
Answer:	Yes: the "superhero" vertex in the bottom right.

Question:	Is there a Tutte set in Figure 16.1a?
Answer:	Yes, and the simplest is the empty set! (This is allowed, and sometimes it is the only option.)

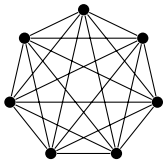
Tutte not only described what we now call Tutte sets, but proved the following theorem showing that they are the only kind of obstacle to be found.

Theorem 16.1 (Tutte's theorem)**.** *Every graph G has a perfect matching if and only if it has no Tutte set.*

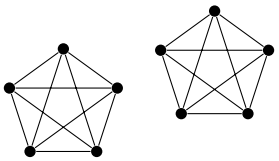
As with Hall's theorem, one direction of Tutte's theorem is much shorter than the other, and the proof of that direction is what motivated our definitions: we defined a Tutte set specifically because we had an argument in mind for why a graph with a Tutte set could not have a perfect matching.

Proof of the "only if" direction of Theorem 16.1. Suppose that U is a Tutte set in a graph G, so that $G-U$ has $k > |U|$ odd components. Let M be a maximum matching in G, and let M' be the largest subgraph of M that is also a matching in $G-U$. Then M' must leave at least k vertices of $G-U$ uncovered: at least one from each odd component. However, M cannot do much better! An edge of M is missing from M' only if it has an endpoint in U, and there are at most $|U|$ such edges; these can cover at most $|U|$ of the vertices of $G-U$ that M' left uncovered. Since $k > |U|$, we know that there are still some vertices of $G-U$ that are uncovered by M; therefore M is not a perfect matching. □

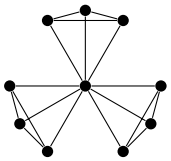
To prove the other direction of Tutte's theorem, we will not use Tutte's original argument, which relied on linear algebra. Instead, we will see an argument from Lovász and Plummer's *Matching Theory*. We will arrive at it indirectly, by first describing a special class of graphs called "saturated graphs" for which the proof will be easier.



(a) K_7



(b) Two copies of K_5



(c) A nonempty Tutte set

Figure 16.2: Several examples of saturated graphs

16.2 Saturated graphs

Call a graph G *saturated*[26] if G has no perfect matching, but if e is any edge not already present in G, then adding e to G would create a perfect matching. (This includes the case of odd complete graphs K_{2n+1}: they are saturated because they do not have a perfect matching, and there is no edge e that can be added.)

Starting from any graph that does not have a perfect matching, we can arrive at a saturated graph by adding edges, one at a time, for as long as this is possible without creating a perfect matching.

Question:	How could we do that in Figure 16.1b?
Answer:	We could add edges to each house to make it a copy of K_5, and then add edges from the extra vertex to every vertex of each house.

Figure 16.2 shows several more examples of saturated graphs. The idea of Tutte sets helps us come up with more: if we have a Tutte set, we can add any edges that don't change its structure.

Question:	If we start with the four houses in Figure 16.1a, and turn each house into a copy of K_5, is the result saturated?
Answer:	No: if we add an edge between two houses, that increases the size of the matching, but it's still not a perfect matching. We can go all the way up to a graph with a copy of K_5 and a copy of K_{15} before it's saturated.

Using Tutte's theorem, we can describe what saturated graphs look like without too much trouble. (Proposition 16.2 does not describe saturated graphs completely, but it will be enough for us.)

Proposition 16.2. *If a graph G is saturated, then for some set of vertices U, G contains every edge with at least one endpoint in U, and every edge between two vertices in the same connected component of $G - U$.*

[26] The word "saturated" is commonly used in graph theory for this type of definition, but it usually needs to be qualified with some other adjectives to say what kind of problem G is saturated for. In this book, we will not need this term outside this chapter, so I will just say "saturated" for brevity, even though it's not as precise.

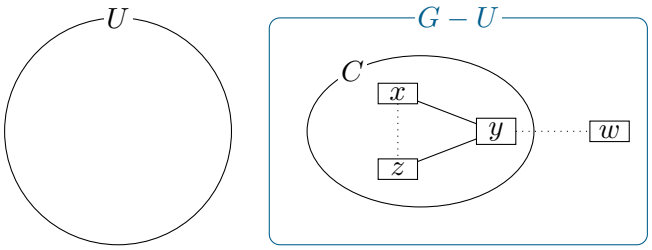


Figure 16.3: Vertices x, y, z, w in the proof of Proposition 16.2; edges xz and yw are not present

Proof of Proposition 16.2 using Tutte's theorem. If a graph G is saturated, then it has no perfect matching, so by Tutte's theorem, it must have a Tutte set U. If an edge e can be added to G that would not change U's status as a Tutte set, then the resulting graph $G + e$ still wouldn't have a perfect matching (again, by Tutte's theorem), which would violate the definition of a saturated graph. Therefore every such edge must already exist in G. Which edges are these?

Well, every edge with at least one endpoint in U is such an edge, because adding it wouldn't affect $G - U$, so U would still be a Tutte set. Also, every edge within a connected component of $G - U$ is such an edge, because adding it wouldn't affect the number of vertices in that component of $G - U$.

Therefore all such edges are already present in G, proving the proposition. □

I made a point to say that this proof used Tutte's theorem, because it means that the proof is not good enough for our purposes. What are those purposes? We are going to reverse the logic, using Proposition 16.2 to prove Tutte's theorem!

16.3 Proof of Tutte's theorem

Let's begin with a fresh proof of Proposition 16.2, so that we can use this proposition in a proof of Tutte's theorem without making the argument circular.

Proof of Proposition 16.2 without using Tutte's theorem. Start by taking U to be the set of all vertices that are adjacent to every other vertex.

Suppose that some connected component C of $G - U$ does not contain every edge it possibly could. Choose $x \in V(C)$ such that not every vertex of C is adjacent to x; let z be a vertex in C at distance 2 from x, and let y be their common neighbor. Because $x, y, z \notin U$, vertex y must not be adjacent to every vertex; there must be a fourth vertex w (potentially outside C) not adjacent to y. This is illustrated in Figure 16.3.

We have two ways to make G bigger: we could add edge xz, or we could add edge wy. Because G is saturated, each of these bigger graphs must have a perfect matching; because G has no perfect matching, those two matchings must each use the added edge. So if we remove the added edge, we see that G has two matchings that are nearly perfect: a matching M_{xz} that covers all vertices except x and z, and a matching M_{wy} that covers all vertices except w and y.

In Chapter 14, to compare two matchings M and N, we looked at their symmetric difference $M \oplus N$. We will do the same here. We know in general that $M_{xz} \oplus M_{wy}$ consists of cycles and

alternating paths: paths that alternate between edges of M_{xz} and edges of M_{wy}. In this case, we can say even more.

Question:	Where can the alternating paths in $M_{xz} \oplus M_{wy}$ start and end?
Answer:	Only at the vertices x, y, z, and w. All other vertices have degree 1 in both M_{xz} and M_{wy}, so they have degree 2 or 0 in $M_{xz} \oplus M_{wy}$.

So $M_{xz} \oplus M_{wy}$ contains an M_{wy}-alternating path that starts at vertex w and ends at one of the three vertices x, y, or z. All three options are good for us:

- If it ends at y, then it's an M_{wy}-augmenting path, since it starts and ends at an uncovered vertex.
- If it ends at x or z, then we can follow up with edge xy or zy to go to y; again, we get an M_{wy}-augmenting path.

Question:	If we follow up a $w - x$ alternating path by going from x to y, why is that path still alternating?
Answer:	We must have arrived to x by an edge of M_{wy}, because M_{xz} leaves x uncovered. Meanwhile, xy is not an edge of M_{wy}, because M_{wy} leaves y uncovered.

By Lemma 14.3, we can use the M_{wy}-augmenting path to improve M_{wy} to a perfect matching in G. But we assumed G was saturated, and didn't have a perfect matching! This is the contradiction that finishes the proof. □

Let me explain the reasons behind what's happened so far. It's common that when proving a theorem about graphs that don't have a property (such as a perfect matching), it is enough to prove the theorem about graphs that don't have the property, but have as many edges as possible: in our case, about saturated graphs. We're about to see in a moment that this is true here: proving Tutte's theorem will be much easier for saturated graphs than for graphs with no special properties.

So we want to learn about saturated graphs. Since we suspect that Tutte's theorem is true even before we have a proof, we started by using it to understand what saturated graphs ought to look like. Then, we went back and proved the same result in legitimate ways, so that it's no longer circular reasoning to use it to prove Tutte's theorem.

Now let's see if our efforts have paid off!

Proof of the "if" direction of Theorem 16.1. Let G be a graph that does not have a perfect matching.

Question:	How can we finish the proof if G has an odd number of vertices?
Answer:	In this case, $U = \varnothing$ is a Tutte set in G, so Tutte's theorem holds.

So we assume that G has an even number of vertices. Then the reason that G doesn't have a perfect matching is simply because it's missing some edges.

Greedily add edges to G, one by one, for as long as this does not create a perfect matching, until we cannot add edges any longer. The result is a saturated graph H that contains G as a spanning subgraph.

By Proposition 16.2 applied to H, there is some set of vertices U such that H contains every edge with at least one endpoint in U, and every edge with both endpoints in the same connected component of $H - U$. We will first prove that U is a Tutte set in H, and then that it is a Tutte set in G.

Let M be a maximum matching in $H - U$. In every connected component C of $H - U$, all edges are present, so the only possible reason why M might not cover every vertex of C is that C is an odd component. What's more, if there are $|U|$ or fewer odd components, then we can extend M to a perfect matching of H: match vertices in U to vertices outside U not covered by M, and if any vertices in U are left, match them to each other. So there must be more than $|U|$ odd components in $H - U$, which exactly means that U is a Tutte set in H.

Question:	Where did we need the assumption that G and H have an even number of vertices?
Answer:	Otherwise, "if any vertices in U are left, match them to each other" might not work: we might end with one vertex uncovered.

Since G is a subgraph of H, every connected component of $H - U$ breaks down into one or more components of $G - U$. However, an odd number cannot be the sum of many even numbers; therefore every odd component of $H - U$ must contain at least one odd component of $G - U$. We conclude that $G - U$ has at least $|U|$ odd components, which means that U is also a Tutte set in G. □

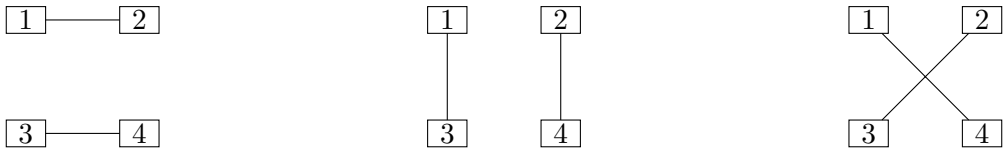
16.4 1-factorizations

A practical application of matchings in graphs that we have yet to consider is tournament design. (We will only explore the basics of the overlap between this field and graph theory.)

Suppose that we would like to organize a round-robin chess tournament[27] between n players. A single chess game can take a while, so we want to schedule as many games at the same time as possible. When n is even, we can always begin by dividing n people into $n/2$ arbitrary pairs and scheduling a game between each pair. This is a perfect matching in K_n, which is not a very difficult matching problem.

However, this only tells us what to do in the first round; future rounds are more difficult, because we don't want to repeat any games. If the first round is a perfect matching M_1 in K_n, then we would like the second round to be a perfect matching M_2 in the complement $\overline{M_1}$, the

[27] "Round-robin", in case you're unfamiliar with the terminology, means that every participant plays a game against every other participant.



(a) The 1-factor $(x_1x_2)(x_3x_4)$ (b) The 1-factor $(x_1x_3)(x_2x_4)$ (c) The 1-factor $(x_1x_4)(x_2x_3)$

Figure 16.4: A 1-factorization of K_4: three disjoint 1-factors whose product is $x_1^3x_2^3x_3^3x_4^3$

third round to be a perfect matching in $\overline{M_1 \cup M_2}$, and so on. These matching problems can easily become impossible if we schedule rounds of the tournament with no foresight.

For example, with 6 players, we hope to finish in 5 rounds, because each player has 5 opponents to face. However, if our first three matchings are poorly chosen, then they might leave us with a graph that has no perfect matching of its own. We would have to ask two people to sit out of the fourth round, and then we would have to schedule 6 rounds total.

Question:	Can this problem really occur? How do we run a 6-player tournament badly?
Answer:	Taking $V(K_6)$ to be $\{1, 2, 3, 4, 5, 6\}$, we might mistakenly begin with three matchings that all match even numbers to odd numbers: for example, $\{12, 34, 56\}$, then $\{14, 25, 36\}$, then $\{16, 23, 45\}$. Now, the remaining graph has two odd components and no perfect matching.

Instead of solving this problem one matching at a time, we will have to solve it all at once: we want to find a decomposition of G, writing at as a union of perfect matchings that share no vertices. There is a special term for this kind of decomposition.

Definition 16.1. *A* **1-*factorization*** *of a graph G is a decomposition of G into perfect matchings: a representation*

$$G = M_1 \cup M_2 \cup \cdots \cup M_k$$

where each M_i is a perfect matching, and each edge of G appears in exactly one M_i.

The term "1-factorization" goes back to the early days of graph theory, where the idea of a graph was more algebraic: an edge xy was really thought of as the product of two variables x and y, and a graph was just the product of such edges. For example, a cycle with vertices $\{x, y, z\}$ would be the expression $(xy)(xz)(yz)$. Some other modern terms have their origin in those days. For example, if you simplify the product that we use to represent our cycle, you get $x^2y^2z^2$; the degree of each variable (in the algebraic sense) is exactly the degree of the corresponding vertex (in the graph-theoretic sense)!

Viewed from this point of view, a 1-factorization really is a factorization of the graph: a way to write it as a product of factors in which every variable has degree 1. (For this reason, perfect matchings are also sometimes called 1-factors.) For example, the graph K_4, viewed as the product $(x_1x_2)(x_1x_3)(x_1x_4)(x_2x_3)(x_2x_4)(x_3x_4)$, has the 1-factorization

$$\underbrace{(x_1x_2)(x_3x_4)}_{\text{1-factor}} \cdot \underbrace{(x_1x_3)(x_2x_4)}_{\text{1-factor}} \cdot \underbrace{(x_1x_4)(x_2x_3)}_{\text{1-factor}}.$$

Figure 16.4 shows this factorization in a more modern way.

Of course, what's important is finding the 1-factorization, not representing it.

Theorem 16.3. *The complete graph K_n has a 1-factorization whenever n is even.*

Proof. It is much easier to draw a diagram of the 1-factorization than it is to give a diagram-free proof. Begin with a diagram of K_n that places $n-1$ of the vertices at evenly spaced points around a circle, and the last vertex in the middle of the circle. For the i^{th} matching M_i, take the edge between the middle vertex to the i^{th} vertex around the circle, as well as all edges perpendicular to this edge in the diagram.

(Figure 16.5 shows an example of this construction in the case $n = 8$.)

In a way, the diagram also leads to a geometric proof. Name the $n-1$ matchings after the $n-1$ radial edges. For each edge xy between two outer vertices, construct the diameter perpendicular to xy, and one half of that diameter will be an edge from the middle vertex. This tells us which matching contains edge xy; in particular, it tells us that xy is in exactly one matching. To see that it's a matching, we first check that xy does not share a vertex with the radial edge perpendicular to it (the lines intersect at the midpoint of edge xy, not at a vertex). In all other cases, two edges xy and $x'y'$ in the same matching are parallel: they do not share a vertex because, as lines, they do not intersect.

For a diagram-free proof, we rely on modular arithmetic instead. Number the vertices 0 through $n-1$. For each $i = 0, 1, \ldots, n-2$, define the matching M_i to contain the edge $\{i, n-1\}$ as well as all the edges $\{i-k \bmod n-1, i+k \bmod n-1\}$ for $k = 1, \ldots, n/2-1$. Since the $n-1$ values

$$i-(n/2-1), i-(n/2-2), \ldots, i-1, i, i+1, \ldots, i+(n/2-1)$$

are distinct modulo $n-1$, no vertices are repeated, and therefore M_i really is a matching.

Next, we show that no edge is contained in multiple matchings. This is true for edges incident to $n-1$, since each matching is defined to contain a different one of these edges. Otherwise, take an edge $xy \in E(M_i) \cap E(M_j)$. Since $xy \in E(M_i)$, we can write it as

$$\{x, y\} = \{i-k \bmod n-1, i+k \bmod n-1\}$$

for some k, so $x+y \equiv (i-k)+(i+k) = 2i \pmod{n-1}$. Similarly, since $xy \in E(M_j)$, then $x+y = 2j \pmod{n-1}$. But n is even and $0 \le i, j \le n-2$, so $2i \equiv 2j \pmod{n-1}$ can only occur if $i = j$.

Question:	What goes wrong at this step if n is odd?
Answer:	Then $2i \equiv 2j \pmod{n-1}$ really is possible for two values i and j. For example, if $n = 7$, we'd be working modulo 6, and $2 \cdot 1 \equiv 2 \cdot 4 \pmod 6$.

From the previous paragraph, it also follows that every edge is contained in some matching. If there are $n-1$ matchings, and each contains $n/2$ edges, then together they contain $\frac{n(n-1)}{2}$ edges total, and we've shown that there's no overlap. But there are only $\frac{n(n-1)}{2}$ edges in K_n, so we've included them all. □

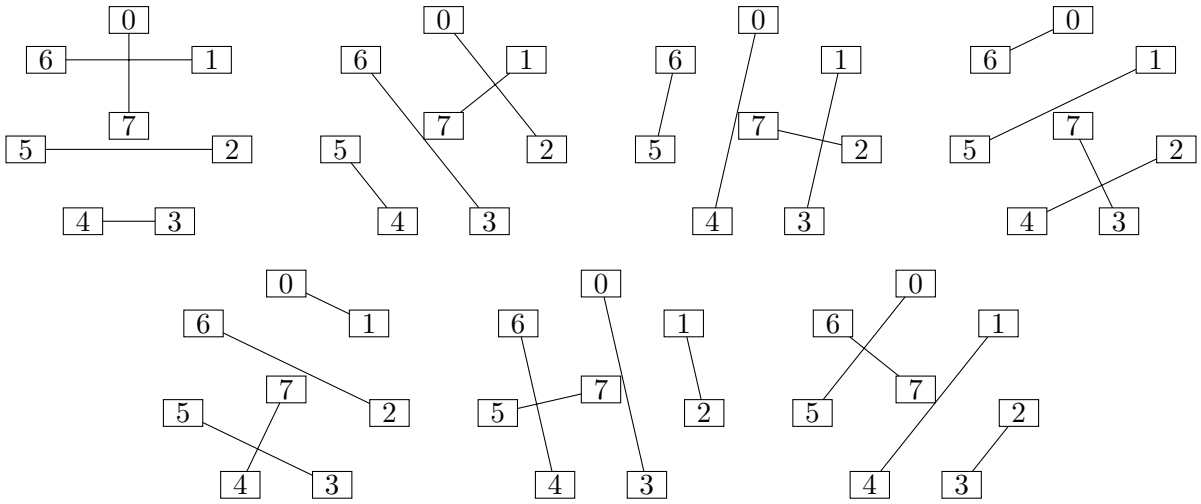


Figure 16.5: The $n = 8$ case of Theorem 16.3

In my opinion, the geometric proof is the "real" reason that Theorem 16.3 is true, but I have included the number-theoretic proof for completeness. It requires some background in number theory to follow, but in reality, working modulo $n-1$ is just a diagram-free way to put $n-1$ evenly spaced points around a circle.

Now we know how to schedule a round-robin tournament between n people in just $n-1$ rounds, if n is even!

Question:	What do tournament organizers do if n is odd?
Answer:	In each round, one player gets a "bye" and sits out. This is equivalent to scheduling an $(n+1)$-player tournament with one player named "bye" who doesn't really exist.

Since $n+1$ is even whenever n is odd, adding a fictional player named "bye" reduces the problem to a case where Theorem 16.3 applies. Of course, with $n+1$ players, we require n rounds, even if one of the players is fictional. This proves the following corollary:

Corollary 16.4. *When n is odd, the complete graph K_n has a decomposition into n matchings which are each nearly perfect (covering $n-1$ of the n vertices).*

Many other graphs can be shown to have 1-factorizations. Let's briefly return to bipartite graphs to prove one more result, originally also due to Kőnig [64]:

Theorem 16.5. *Every regular bipartite graph has a 1-factorization.*

Proof. By Theorem 15.2, every regular bipartite graph G has a perfect matching M. If G is k-regular, then $G - E(M)$ is $(k-1)$-regular: every vertex of G is incident to one edge of M, so its degree goes down by 1 in $G - E(M)$.

Repeat this argument, removing perfect matchings from G until it is 0-regular, and there are no more edges. (Formally, this proof should be rephrased as an induction on k; can you see how?) The perfect matchings removed at each step form a 1-factorization of G. □

16.5 Increasing walks

The problem of finding a 1-factorization of K_n was historically first studied by tournament organizers, not graph theorists. However, that does not mean it is not useful in graph theory. I encountered the following problem as a graduate student.

Take the complete graph K_n, and make it into a weighted graph by giving each edge e a weight $c(e)$; in this problem, we will insist that all the weights should be different. A walk $(x_0, x_1, x_2, \ldots, x_l)$ is called an *increasing walk* if the weights go up along the walk: if

$$c(x_0x_1) < c(x_1x_2) < \cdots < c(x_{l-1}x_l).$$

What is the longest possible increasing walk? Well, it depends on the labels. We can have very long increasing walks if the weights cooperate. Suppose that before choosing edge weights, we pick a walk $(x_0, x_1, x_2, \ldots, x_l)$ that we'd like to be increasing. Provided the walk does not repeat any edges, we can make it so! Simply set $c(x_{i-1}x_i) = i$ for $i = 1, 2, \ldots, l$.

Question:	What is the longest walk in K_n that does not repeat any edges?
Answer:	If n is odd, then all degrees are even, so K_n has an Euler tour: a walk of length $\binom{n}{2}$. If n is even, then we can delete any matching and be left with an Eulerian graph, with an Euler tour of length $\binom{n}{2} - n/2$.

This is a best-case analysis, and the short solution to it shows us why worst-case analyses are much more interesting. Instead, let's ask the question: what is the longest possible length of an increasing walk that we can guarantee, no matter what the weights of the edges are?

This problem was first studied by Ron Graham and Daniel Kleitman, who proved in 1973 [39] that it was possible to find weighted graphs in which the longest increasing walk is much shorter.

Proposition 16.6. *For all even n, there is a weighted complete graph on n vertices in which no increasing walk has length more than $n - 1$.*

Proof. Use Theorem 16.3 to decompose K_n into $n-1$ perfect matchings M_1 through M_{n-1}. For each i, and each $e \in E(M_i)$, set $c(e) = i$. Okay, that doesn't quite work, because the weights all have to be different, but it will have the same effect if $c(e)$ is any number in the interval $[i, i + \frac{1}{2}]$: we will never end up comparing two edges in a matching anyway.

An increasing walk in this weighted graph cannot use more than one edge from any matching M_i. After taking its first edge from that matching, it cannot immediately take a second edge, because no two edges in M_i share an endpoint. So the walk must follow up by going to a different matching: since the walk is increasing, it must select an edge from M_j, for some $j > i$. But this

edge has a greater weight than any edge of M_i, so the increasing walk can never return to M_i again.

Since there are only $n-1$ matchings, the increasing walk cannot use more than $n-1$ edges. □

Graham and Kleitman proved more than this. They generalized Proposition 16.6 to work for odd n as well, excluding the special case $n = 3$ and $n = 5$. Moreover, they proved that this upper bound is always achievable! I will present their solution in a different style, as described by Peter Winkler [108] and attributed to Ehud Friedman.

Proposition 16.7. *In every weighted complete graph on n vertices, there is an increasing walk of length at least $n-1$.*

Proof. Imagine that we put n different people on the n vertices of the complete graph. Then, we call out the edges of the graph one by one, in increasing order of weight. Whenever we call out an edge xy, the two people standing on vertices x and y trade places, walking along edge xy in opposite directions.

At the end, we will ask each person to describe the walk they took. All these walks must be increasing, because all edges were announced in increasing order. Let $l_1, l_2, \ldots, l_n$ be the lengths of the n walks.

Question:	How can we express the sum $l_1 + l_2 + \cdots + l_n$ in another way?
Answer:	The sum counts each edge of the graph twice, since two people walk each edge, so it is equal to $2\|E(K_n)\|$, which is $n(n-1)$.

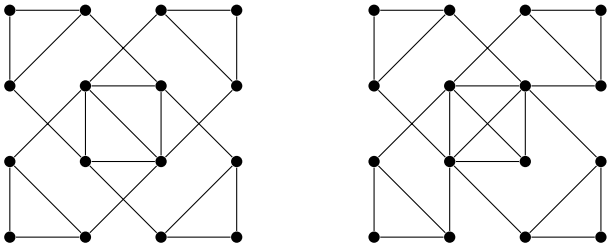
If the lengths $l_1, l_2, \ldots, l_n$ add up to $n(n-1)$, then their average is

$$\frac{l_1 + l_2 + \cdots + l_n}{n} = \frac{n(n-1)}{n} = n - 1,$$

and it is impossible for all n lengths to be below average. Therefore at least one walk must have length at least $n-1$. □

16.6 Practice problems

1. In one of the graphs below, find a perfect matching. In the other, prove that there is no perfect matching, by finding a Tutte set.



2. Theorem 15.2 from the previous chapter implies that every 3-regular bipartite graph has a perfect matching.

   Prove that the word "bipartite" cannot be left out: give an example of a 3-regular graph which is not bipartite, and does not have a perfect matching.

3. (USAMO 1989) The 20 members of a local tennis club have scheduled exactly 14 two-person games among themselves, with each member playing in at least one game. Prove that within this schedule there must be a set of 6 games with 12 distinct players.

4. Use Tutte's theorem to prove Hall's theorem.

5. Let $o(G)$ denote the number of odd components in a graph G. Prove that if G is a graph and U is a subset of $V(G)$, then

$$\alpha'(G) \leq \frac{1}{2}\big(|V(G)| - o(G-U) + |U|\big).$$

   (More is true. The Tutte–Berge formula is a generalization of Tutte's theorem saying that the two sides of this inequality are equal for some set U.)

6. Prove that if G is a graph with an even number of vertices and no perfect matching, then it has a Tutte set S with $o(G-S) \geq |S| + 2$.

7. Suppose that $2n$ people from two different n-person teams participate in a tournament. Describe how to schedule n rounds of simultaneous games between players on opposing teams, so that each person plays once against everyone on the other team.

   (In other words, find a 1-factorization of the complete bipartite graph $K_{n,n}$. Theorem 16.5 assures us that one exists, but doesn't say what it is.)

8. Let G be a copy of K_8 with vertices $\{000, 001, \ldots, 111\}$ (just like the vertices of the cube graph Q_3).

   a) For each nonempty subset $S \subseteq \{1,2,3\}$, let M_S be the matching consisting of all edges in G whose endpoints differ in the positions numbered by S. For example,

$$E(M_{\{1,2\}}) = \Big\{\{000,110\},\{001,111\},\{010,100\},\{011,111\}\Big\}.$$

   Prove that the seven matchings $M_{\{1\}}, M_{\{2\}}, M_{\{3\}}, M_{\{1,2\}}, M_{\{1,3\}}, M_{\{2,3\}}, M_{\{1,2,3\}}$ are a 1-factorization of G.

   b) Prove that this 1-factorization is not isomorphic to the one found in Theorem 16.3. That is, prove that it is not possible to replace the labels $\{000, 001, \ldots, 111\}$ by $\{0, 1, \ldots, 7\}$ in any way to turn this 1-factorization into the one shown in Figure 16.5.

9. Define an increasing path to be a path such that one of the walks representing it is an increasing walk.

   We can prove a version of Proposition 16.7 for increasing paths by changing the rules slightly in that proof. Whenever edge xy is called out, if the person on x has already visited y, or the person on y has already visited x, then both people stay put. This ensures that at the end, each person's walk represents an increasing path.

a) Suppose that each person walks a path of length at most L. Prove that each person is responsible for "rejecting" at most $\frac{L(L-1)}{2}$ edges.

b) At the end, each of the $\binom{n}{2}$ edges was either walked by two people or rejected by at least one person. Use this to prove the inequality $n \cdot \frac{L}{2} + n \cdot \frac{L(L-1)}{2} \geq \binom{n}{2}$.

c) Prove $L \geq \sqrt{n-1}$.

Graham and Kleitman proved a similar result, but unlike the problem of increasing walks, the exact worst-case answer is still not known in the case of increasing paths.

10. A graph G is called claw-free if it does not have a copy of the star graph S_4 as an induced subgraph. In other words, no vertex $x \in V(G)$ can have three neighbors y_1, y_2, y_3 with no edges between them.

Sumner's theorem [95] states that every connected claw-free graph with an even number of vertices has a perfect matching. Prove this using Tutte's theorem.

(Here is a hint for one possible solution: choose a careful ordering $x_1, x_2, \ldots, x_k$ of the vertices in S, then prove by induction on i that the number of connected components of $G - S$ with a neighbor in $\{x_1, x_2, \ldots, x_i\}$ cannot exceed $i + 1$.)

Hard Problems

17 Hamilton cycles

The purpose of this chapter

This chapter begins with many examples of analyzing concrete graphs to find Hamilton cycles, or prove that none exist. The second half of this chapter is more theoretical. If you are reading this book on your own, you should feel free to skip to the section on tough graphs as soon as you feel like you've had enough of the discussions that come before it. If you are teaching from this book, then you can give more of the concrete examples for a gentle introduction, or leave your students to read them on your own if your goal is to set a faster pace.

The problem of identifying Hamiltonian graphs is the first of the "hard problems" we will consider in this book. What do I mean by this? Well, most precisely put, it is an issue of complexity theory: all the "hard problems" covered in this part of the textbook are so-called NP-complete problems. You may have heard of the "P versus NP problem", an open problem in computer science whose solution is worth $1 000 000. Finding an efficient algorithm to solve an NP-complete problem, or proving that such an algorithm does not exist, would earn you that million-dollar prize!

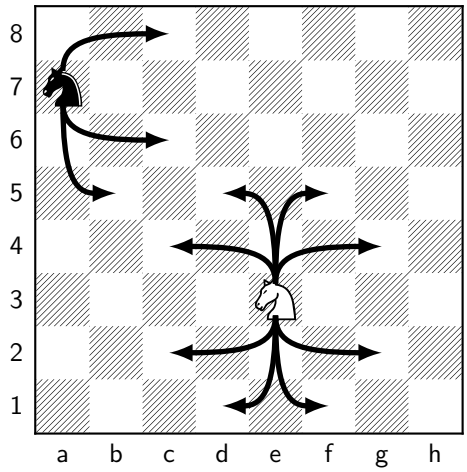
This is not a textbook on complexity theory, however, so I will not go into the details. For us, the consequence of tackling hard problems is that we will not be able to solve them efficiently, or prove theorems with simple if-and-only-if conditions describing when they have a solution. Instead, we will prove partial guarantees, lower and upper bounds, and discuss algorithms that only sometimes work.

17.1 Knight's tours

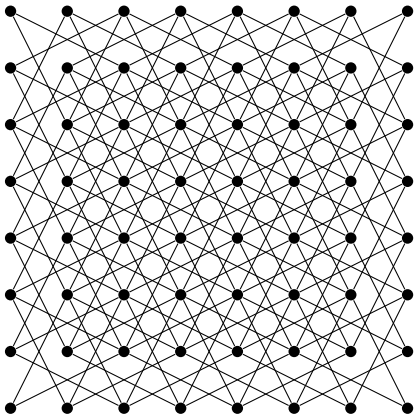
A knight is a chess piece that moves either by taking one step vertically and two steps horizontally, or by taking two steps vertically and one step horizontally, as shown in Figure 17.1a. A knight in the middle of the board can move to a total of 8 different squares, though this number is reduced if the knight is near the edges of the board.

For the purpose of graph theory, Figure 17.1b is the correct way to represent how a knight moves in chess. This is the 64-vertex graph with a vertex for each square of the chessboard, where two vertices are adjacent if a knight can move from one square to the other. In this chapter, we will just call this graph the 8×8 *knight graph*.

A knight's tour is a sequence of knight moves that visits every square of the chessboard exactly once. Finding a knight's tour is a famous mathematical chess puzzle. A slightly harder variant is the problem of finding a closed knight's tour: here, after visiting every square, the knight must make one final move and end at the square where it started. Both problems have been studied extensively long before graph theory, both with generic methods and ones specialized to



(a) How a knight moves in chess



(b) The graph of knight moves

Figure 17.1: Chess, knights, and graph theory

the chessboard [6]. For example, in 1832, H. C. von Warnsdorff proposed the short rule, "Start anywhere, and always move to the space from which the number of moves to unvisited squares is smallest." This is quite likely (though not certain) to produce a knight's tour; it is also a reasonable heuristic for the general problem we are about to study.

Graph-theoretically, the knight's tour problem is the problem of finding a path or cycle in the 8×8 knight graph that contains every vertex of the graph. In general, we say:

Definition 17.1. *A **Hamilton path** (or spanning path) in a graph is a path that passes through every vertex of the graph. A **Hamilton cycle** or (or spanning cycle) in a graph is a cycle that passes through every vertex of the graph.*

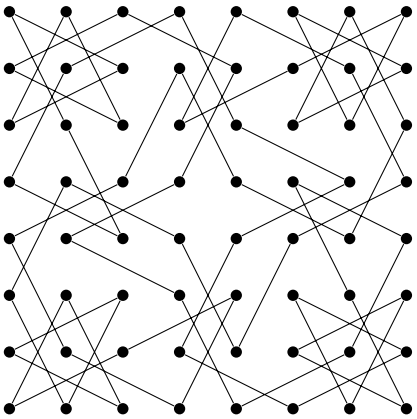
Usually, Hamilton cycles are considered to be the more fundamental object in graph theory, because they have a symmetry that paths lack: every vertex plays an identical role. As a result, graphs with Hamilton cycles are the ones we give a special name.

Definition 17.2. *A graph is called **Hamiltonian** if it has a Hamilton cycle.*

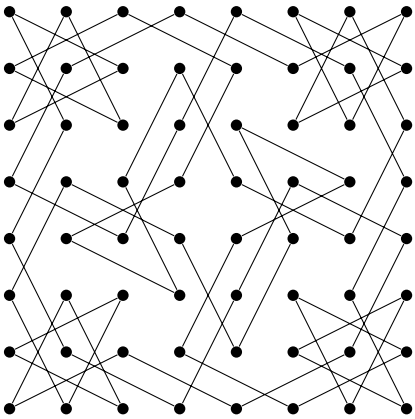
(Actually, there is also a term for graphs that have a Hamilton path: they are called traceable. This term is much less commonly used.)

Figure 17.2 shows two interesting subgraphs in the 8×8 knight graph. Both of these were found by Dan Thomasson [96], whose website features an impressive collection of knight's tours.

Question:	However, only one of the diagrams in Figure 17.2 shows a Hamilton cycle. Which one, and what's wrong with the other one?
Answer:	Figure 17.2b shows the Hamilton cycle. In Figure 17.2a, every vertex has degree 2, but the edges don't form a single cycle: there are two connected components!

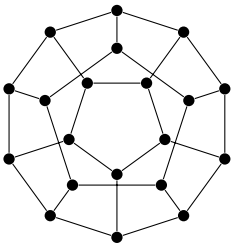


(a) Is this a Hamilton cycle?

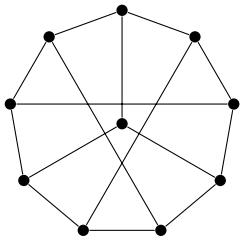


(b) Is this a Hamilton cycle?

Figure 17.2: Two symmetric subgraphs of the 8×8 knight graph



(a) The dodecahedron graph



(b) The Petersen graph

Figure 17.3: Are these graphs Hamiltonian?

An object such as the one shown in Figure 17.2a is called a 2-*factor*: a spanning subgraph (in this case, of the 8×8 knight graph) in which every vertex has degree 2. Every Hamilton cycle is a 2-factor, but not vice versa: a Hamilton cycle must also be connected!

In Chapter 16, we saw the term 1-*factor* used as a synonym for a perfect matching: a spanning subgraph in which every vertex has degree 1. Algorithmically, a 2-factor is not much harder to find than a perfect matching. On the other hand, a Hamilton cycle is much harder to find. There is no known efficient algorithm, and no characterization like Tutte's theorem (Theorem 16.1) of when a Hamilton cycle exists.

Question:	Does the 8×8 knight graph have a 1-factor, or perfect matching?
Answer:	Yes; one way to be sure without doing any additional work is that we can take every other edge in the knight's tour shown in Figure 17.2b, and get a perfect matching. You might also be able to find some nicely symmetric perfect matchings from scratch.

17.2 Two examples

Before we develop any general theory, let's consider two examples: the two graphs in Figure 17.3. One of these graphs is Hamiltonian; the other is not. (Try them for yourself before you read my explanation, and see what you think.)

The dodecahedron graph in Figure 17.3a has a historical connection to Hamilton cycles. The mathematician William Rowan Hamilton was a big fan of the dodecahedron; in the 1850s, he invented a puzzle called the "Icosian Game", which was effectively to find a Hamilton cycle in the dodecahedron graph [6]. Although Hamilton's interest in the puzzle was more related to his study of abstract algebra, the Icosian Game is the reason why we refer to spanning cycles as Hamilton cycles.

There is no obvious starting point to the puzzle, so I will give you an "in" with some mathematical trickery.

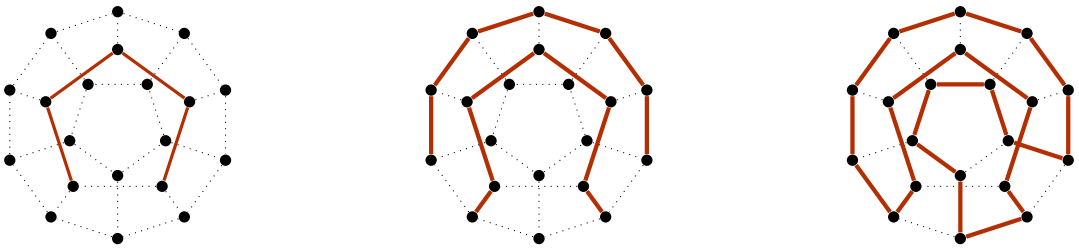
Question:	As a 3-dimensional shape, the dodecahedron has 12 faces, each of which is a regular pentagon. A Hamilton cycle in the dodecahedron graph uses several edges from each pentagon. What is the average number of edges used per pentagon?
Answer:	Since the graph has 20 vertices, a Hamilton cycle has 20 edges. Each edge is also an edge geometrically: an edge between two faces. So if we add up the number of edges of the cycle on each pentagon, we will get 40: each edge will be counted twice. There are 12 pentagons, so the average number of edges per pentagon is $\frac{40}{12} = \frac{10}{3}$.

Question:	Is it possible that there is no pentagon from which we use 4 edges?
Answer:	No. We can't use all 5 edges from a pentagon—then our cycle would end early. But we also can't use only 3 or fewer edges from all pentagons: then every pentagon would be below average, which is impossible.

The pentagons are all alike. (Formally, as in Chapter 2, we should say that there is an automorphism of the dodecahedron graph that can turn any dodecahedron into any other.) So we can arbitrarily decide that the "front" pentagon will be the one in which 4 edges are used. This leads us to the partial cycle in Figure 17.4a.

From here, we can make two other deductions.

Question:	Our partial cycle is really a path with two ends. How should we continue from those ends?
Answer:	We can't use the edge connecting them, because then our cycle would end early. So we have to take the only remaining edges: the ones going outward.



(a) Step 1 (b) Step 2 (c) Step 3

Figure 17.4: Solving the "Icosian Game"

Question:	Which edges incident to the topmost vertex should be part of the cycle?
Answer:	There is no choice here: the vertex below the topmost vertex already has degree 2, so it can't accept any more edges. We must use the two edges going left and right.

Applying the same logic to several other vertices on the perimeter, we arrive at a bigger partial solution: the one shown in Figure 17.4b.

At this point, one more "symmetry breaking" step is necessary. From the bottom vertex in the diagram, we must use either the edge going left or the edge going right. Which one? Well, it can't possibly matter: both the diagram, and our partial solution so far, have left-right symmetry! So we can arbitrarily pick the edge going right to get one possible solution; its mirror image will contain the edge going left, instead.

After this choice, there are several more forced decisions like ones we already made: once we use two edges out of a vertex x, if xy is the third edge, then it cannot be part of the Hamilton cycle, which forces our hand at vertex y. Repeating these steps is all that's necessary to arrive at Figure 17.4c, the complete solution.

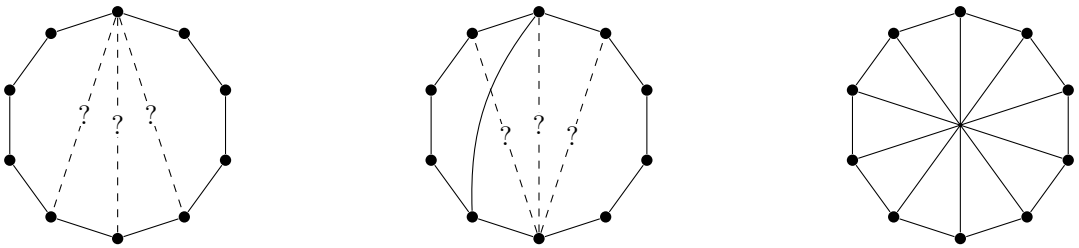
As you see from this example, finding a Hamilton cycle really is more like solving a puzzle than solving a math problem. Although deductions like these can often handle small graphs with enough low-degree vertices, they are not reliable. We might have to make guesses and backtrack if we fail, which is a slow and tedious process.

Rather than attempt such a backtracking solution for the Petersen graph, I will show you a different style of argument.

Proposition 17.1. *The Petersen graph is not Hamiltonian.*

Proof. This proof will rely on just two properties of the Petersen graph. First, it is 3-regular, which can be checked quickly by looking at Figure 17.3b. Second, as we checked in Lemma 5.3, it contains no cycles of length 3 or 4.

Now, we proceed in an unusual way. Rather than starting with the Petersen graph and looking for a Hamilton cycle, let's start with the cycle and look for the Petersen graph!



(a) Step 1 (b) Step 2 (c) Step 3

Figure 17.5: Diagrams for the proof of Proposition 17.1

What do I mean? Well, if there were any Hamilton cycle in the Petersen graph, then we would be able to arrange the vertices around a regular decagon whose edges are the edges of that cycle. This would look like the diagram in Figure 17.5a—with five more edges we haven't placed yet completing the diagram of the 15-edge Petersen graph. For example, from the topmost vertex, exactly one of the three dashed edges must exist, we just don't know which one. (We need a third edge from that vertex to give it degree 3, but any edge other than the three dashed edges would create a short cycle.)

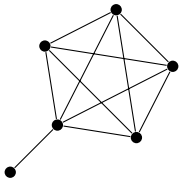
Suppose we pick the left dashed edge. Then we arrive at Figure 17.5b, and we can ask the same question for the bottom-most vertex...

Question:	Which of the three dashed edges in Figure 17.5b can we use, if we want to eventually complete the picture to a diagram of the Petersen graph?
Answer:	None of them! They all create a 3-cycle or 4-cycle.

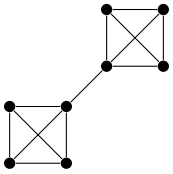
So we cannot pick the left dashed edge (or, by the same argument, the right one). We must pick the edge going directly across. Since there's nothing special about the top vertex, the same argument applies to every vertex: the only way we can hope to obtain the Petersen graph is by joining each vertex to the vertex directly opposite. This gives us the graph in Figure 17.5c.

Question:	Why is the graph in Figure 17.5c not the Petersen graph?
Answer:	It, too, has cycles of length 4: starting from any vertex, take a step across, a step clockwise, another step across, and a step counterclockwise.

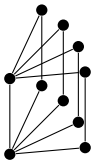
Since all possible attempts to expand a 10-vertex cycle to get a Petersen graph fail, we conclude that the Petersen graph has no 10-vertex cycle as a subgraph: it is not Hamiltonian. □



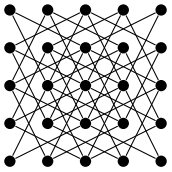
(a) Lollipop graph $L_{5,1}$



(b) 4-barbell graph



(c) Book graph B_4



(d) 5×5 knight graph

Figure 17.6: Four graphs which are not Hamiltonian

17.3 Tough graphs

The proof of Proposition 17.1 is rather long. I can't honestly tell you that the Petersen graph is an unusually bad example; for many graphs, finding a Hamilton cycle or proving that one does not exist is even harder. However, there are also many graphs for which we can give a quick argument to rule out a Hamilton cycle. Four of them are shown in Figure 17.6.

Question:	Why is the lollipop graph in Figure 17.6a not Hamiltonian?
Answer:	It has a leaf: a Hamilton cycle could not possibly enter that vertex and leave it.

Question:	Why is the barbell graph in Figure 17.6b not Hamiltonian?
Answer:	The edge joining the two copies of K_4 is a bridge; by Lemma 9.2, it is not part of any cycles. But without using it, a cycle can't contain vertices from both copies of K_4.

Question:	Why is the book graph in Figure 17.6c not Hamiltonian?
Answer:	Between visits to every "page" of the book (the vertices of degree 2), a cycle must pass through at least one vertex of the "spine" (the vertices of degree 5). But there are only 2 vertices in the spine, so a cycle cannot visit all 4 pages.

Question:	Why is the 5×5 knight graph in Figure 17.6d not Hamiltonian?
Answer:	Remember the black-and-white covering of a chessboard? On a 5×5 chessboard, there would be 13 squares of one color, and 12 on the other. A knight always moves to squares of a different color (put differently, the knight graph is bipartite), so a cycle will visit an equal number of squares of each color: it can't visit all 25 vertices.

Surprisingly, all four arguments are special cases of one result!

We say that a graph G is *tough* if, for any $k \geq 1$, deleting k vertices from G results in at most k connected components. You might think, from this definition, that proving that a graph is

tough is a headache involving lots of casework: there are too many ways to delete some number of vertices from a graph! In general, you'd be right.

In some cases, it's not so bad. Here is a lemma that proves that the cycle graph C_n is tough without any casework at all—and, as a bonus, it comes with a mini-review of Chapter 10.

Lemma 17.2. *For all $n \geq 3$, the cycle graph C_n is tough.*

Proof. The cycle graph C_n has only one cycle: the entire graph. So as soon as we delete any vertices from it, we get a graph with no cycles, which we know as a forest. Specifically, if we delete k vertices from C_n, we get an $(n-k)$-vertex forest. Each deleted vertex has degree 2 in C_n, so it costs us at most two edges. Therefore the forest we are left with has at least $n-2k$ edges.

Question:	Why do I say "at most" and "at least"; shouldn't every vertex we delete cost us both edges incident to that vertex?
Answer:	We might delete both endpoints of an edge, in which case we shouldn't count the edge as removed twice.

By a result from much earlier in this book, Proposition 10.3, the number of connected components in a forest is the number of vertices minus the number of edges. In this case, this is at most $(n-k)-(n-2k)=k$. Therefore deleting k vertices results in a graph with at most k components, proving the lemma. □

Starting at C_n is useful, because it gives us a connection to Hamiltonian graphs. This is a result of Václav Chvátal, who defined tough graphs in 1973 for this very purpose [16].

Corollary 17.3. *All Hamiltonian graphs are tough.*

Proof. Every n-vertex Hamiltonian graph G contains a copy of C_n as a subgraph: that's what a Hamilton cycle is! We can think of G as this Hamilton cycle, together with some extra edges it didn't use.

By Lemma 17.2, if we delete k vertices from the Hamilton cycle, the result will have at most k connected components. What happens if we delete those same vertices from G? Those same connected components will still be connected, but it's possible that the extra edges will join some of them together into larger components. This can only reduce the number of connected components, so there are still at most k; therefore G is tough. □

The most profitable way to use Corollary 17.3 is in the form of its contrapositive: if a graph is not tough, then it is not Hamiltonian. If a graph is not tough, then there is a short proof of that fact: simply say which k vertices must be deleted to leave $k+1$ or more connected components. (It is a short proof, not an easy proof, because finding those k vertices may be hard.)

For example, none of the graphs in Figure 17.6 are tough—and the sets of vertices we must delete to prove this are closely related to our earlier arguments proving that none of these graphs are Hamiltonian. This unifies our previous arguments into a single idea.

Corollary 17.3 is a necessary condition: to be Hamiltonian, a graph must be tough. However, it is not sufficient, so we cannot always rely on it. The Petersen graph—a thorn in the side of every graph theorist—is a counterexample.

Proposition 17.4. *The Petersen graph is tough.*

Proof. As we saw in Chapter 5, the Petersen graph has many automorphisms, and in particular, it has an automorphism taking any vertex to any other vertex. Due to this symmetry, it's enough to see what happens when the vertices we delete from the Petersen graph include the vertex at the center of Figure 17.3b, which we will call x.

If we delete x alone, the remaining graph is connected, so the definition of a tough graph holds so far. What's more, we can see from the diagram in Figure 17.3b that the remaining graph is Hamiltonian: we get a Hamilton cycle by going around the perimeter of the diagram. By Corollary 17.3, this graph is tough. So if we delete x and k more vertices from the Petersen graph, we're left with at most k connected components. Since we've deleted $k+1$ vertices, that's even slightly stronger than what we need to conclude that the Petersen graph is tough. □

17.4 Sufficient conditions

It is also possible to find conditions for a graph to be Hamiltonian that are sufficient, but not necessary. That is, if a graph G satisfies these conditions, we can be sure that it has a Hamilton cycle; however, if G does not satisfy these conditions, we learn nothing.

What's the point, then? Well, we can look for conditions that are easy to check. When faced with a concrete problem, we can begin by checking a few such conditions, and if we're lucky, this will tell us that G is Hamiltonian right away. Otherwise, we may have no choice but to embark on a tedious journey full of backtracking.

A promising place to start looking for such conditions is the degree sequence of G. Computing the degree of every vertex may take some time, if the graph is large, but it is not a computationally challenging problem: we can even do it by hand, just by looking at a diagram. At the same time, the degrees of the vertices of G tell us a lot about the structure of G, so we can hope to learn something about Hamilton cycles in G, as well.

Historically, the first such result was a 1952 theorem by Gabriel Dirac [24] (the son of Paul Dirac, the quantum physicist). More results followed, and eventually, the essence of the proof was distilled into a rather peculiar statement, proven by John Bondy and Václav Chvátal in 1976 [8].

Theorem 17.5. *In a graph G with n vertices, let s and t be two non-adjacent vertices with $\deg_G(s) + \deg_G(t) \geq n$. If $G + st$ is Hamiltonian, then so is G.*

Question:	Is it possible that G is Hamiltonian, but $G + st$ is not?
Answer:	No: adding an edge can never destroy a Hamilton cycle. We could have stated Theorem 17.5 as an if-and-only-if condition: $G + st$ is Hamiltonian if and only if G is Hamiltonian.

What is going on here? Well, there are many works of fiction in which the hero is granted special powers by a magic item. Later in the story, the magic item is lost, and the hero is distressed—only to discover that the item wasn't necessary to have the special powers. The magic was really inside the hero the whole time! The statement of Theorem 17.5 is similar: the graph G is granted the special power of being Hamiltonian by the magic edge st. However (provided that edge st satisfies the degree condition) the graph G didn't need that edge: the Hamilton cycle was really inside G the whole time.

Theorem 17.5 tries to help us by giving us a different problem to solve: instead of checking whether G is Hamiltonian, we can check whether $G + st$ is Hamiltonian.

Question:	How can this possibly help us?
Answer:	Well, $G + st$ has one more edge, so finding a Hamilton cycle in $G + st$ might be easier. A single edge might not make a difference, but if we apply Theorem 17.5 repeatedly, we can hope to turn a graph with a well-hidden Hamilton cycle into a graph with many blindingly obvious Hamilton cycles.

Let's prove the theorem first, though.

Proof of Theorem 17.5. Take a Hamilton cycle in $G + st$. If edge st is not part of that cycle, we're done: it also exists in G. If edge st is part of that cycle, then deleting that edge leaves an $s - t$ Hamilton path in G. Let this path be represented by the walk $(x_1, x_2, \ldots, x_n)$ where $x_1 = s$ and $x_n = t$.

Our goal is to find an alternate way to complete the path P to a Hamilton cycle in G, without using edge st. All we really know about G is that, because $\deg_G(s) + \deg_G(t) \geq n$, there must be many other edges incident to s or t. (There might be other edges, too, but we cannot count on them.) In short, the subgraph of G we have available to work with might look something like the diagram in Figure 17.7a.

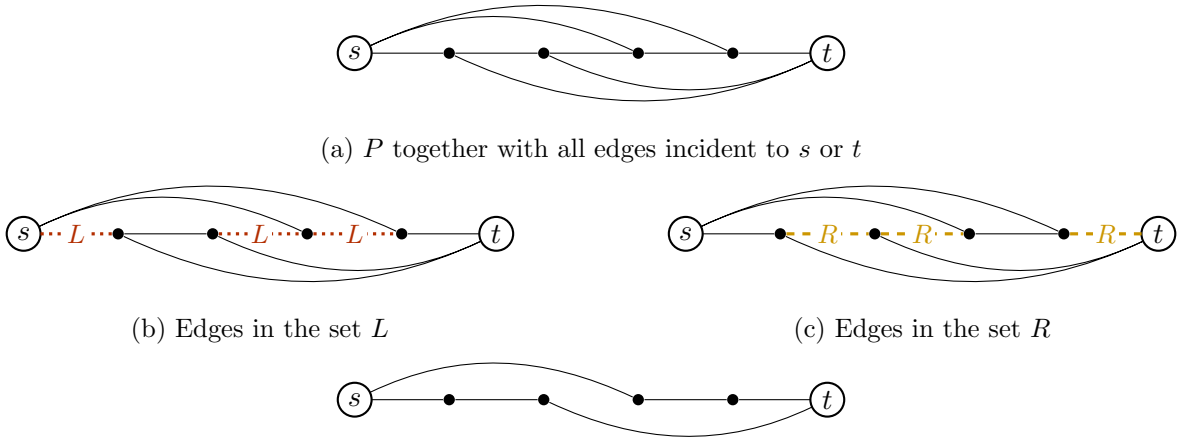
For every edge incident to either s or t, something special happens: we obtain a different Hamilton path, which uses that edge, but leaves out one edge of P. Here are the two ways this can happen:

- If $sx_i \in E(G)$ for some i, then add edge sx_i to P and delete edge $x_{i-1}x_i$. The resulting graph is an $x_{i-1} - t$ Hamilton path.

  We define the set L (for "left") to be the set of all edges of P that can be left out in this way: $L = \{x_{i-1}x_i : sx_i \in E(G)\}$. These are highlighted in Figure 17.7b.

- If $x_{i-1}t \in E(G)$ for some i, then add edge $x_{i-1}t$ to P and delete edge $x_{i-1}x_i$. The resulting graph is an $s - x_i$ Hamilton path.

  We define the set R (for "right") to be the set of all edges of P that can be left out in this way: $R = \{x_{i-1}x_i : x_{i-1}t \in E(G)\}$. These are highlighted in Figure 17.7c.



(a) P together with all edges incident to s or t

(b) Edges in the set L

(c) Edges in the set R

(d) The cycle in G

Figure 17.7: Turning the $s-t$ path in the proof of Theorem 17.5 into a cycle

Question:	Can we say anything about the sizes of the sets L and R?
Answer:	We have $\|L\| = \deg_G(s)$ and $\|R\| = \deg_G(t)$, because for every edge incident to s, we get an element of L, and for every edge incident to t, we get an element of R.

By themselves, edges in L or in R are not anything special, since we aren't looking for more Hamilton paths. The magic happens if we find $x_{i-1}x_i$ in both sets!

Suppose $x_{i-1}x_i \in L \cap R$, so that edges sx_i and $x_{i-1}t$ are both present. Let C be the graph obtained from P by deleting edge $x_{i-1}x_i$ and adding both sx_i and $x_{i-1}t$, as shown in Figure 17.7d. This graph C is a spanning subgraph of G. It is connected: deleting edge $x_{i-1}x_i$ separated P into two connected components, but both sx_i and $x_{i-1}t$ join two vertices in different components, reconnecting the graph. It is 2-regular: compared to P, the degrees of s and t go up by 1, while all other degrees remain the same. Therefore C is a Hamilton cycle!

Question:	The situation is a bit unusual if $x_1x_2 \in L \cap R$. What happens?
Answer:	This would imply that $x_1x_n = st$ is an edge of G. In principle, this makes a cycle in an even simpler way: without deleting anything, we just add edge st to P. However, Theorem 17.5 assumes that this case does not occur: s and t are not adjacent.

Finding an edge in $L \cap R$ is why the degree condition on s and t is needed. Since $|L| = \deg_G(s)$ and $|R| = \deg_G(t)$, we know that $|L| + |R| \geq n$ from the hypotheses of Theorem 17.5. However, L and R are both subsets of $E(P)$, and an n-vertex path has only $n-1$ edges, so $|L \cup R| \leq n-1$. Since $|L \cup R| < |L| + |R|$, there must be some overlap between L and R; as we've already seen, a single edge in $L \cap R$ proves that G is Hamiltonian, which proves the theorem. □

As an example of applying Theorem 17.5, we will use it to derive Dirac's 1952 theorem on Hamilton cycles, with a sufficient condition based on the minimum degree $\delta(G)$.

Corollary 17.6. *If G has $n \geq 3$ vertices and $\delta(G) \geq \frac{1}{2}n$, then G is Hamiltonian.*

Proof. The minimum degree condition on G guarantees that for every two vertices s and t which are not adjacent, we have

$$\deg_G(s) + \deg_G(t) \geq \frac{1}{2}n + \frac{1}{2}n = n,$$

so the hypotheses of Theorem 17.5 hold. Therefore we can add any edges we like to G without changing the problem: if we get a Hamiltonian graph, then G must have been Hamiltonian all along.

The edges we will choose to add to G are all the edges missing from G (one at a time). Once we're done, we get a graph isomorphic to the complete graph K_n. But K_n is definitely a Hamiltonian graph: you can visit the vertices in any permutation you like, then go back to the start, and this will be a Hamilton cycle. Therefore the graph we started with, G, must also have been Hamiltonian. □

How good is Corollary 17.6? Well, we can compare it to Proposition 4.3, which gives $\delta(G) \geq \frac{n-1}{2}$ as a sufficient condition for G to be connected. As we saw in Chapter 4, that condition is the best possible: if the minimum degree of G were any lower than $\frac{n-1}{2}$, then it would not necessarily be connected.

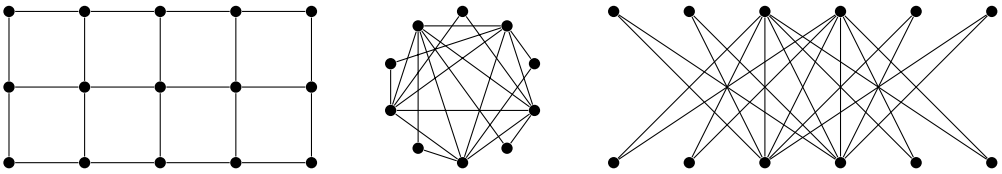
When $\delta(G) = \frac{n-1}{2}$ exactly, we are in the narrow gap between these two results, where the graph is guaranteed to be connected by Proposition 4.3, but not yet guaranteed to be Hamiltonian by Corollary 17.6.

Question:	Are there any n-vertex graphs G with $\delta(G) = \frac{n-1}{2}$ which are not Hamiltonian?
Answer:	Yes: one example is the slightly unbalanced complete bipartite graph $K_{(n-1)/2,(n+1)/2}$. This graph is not Hamiltonian for the same reason that the 5×5 knight graph in Figure 17.6d was not Hamiltonian.

This tells us that the minimum degree condition of Corollary 17.6 is also the best possible: we cannot reduce it any further and still have a true statement.

17.5 Practice problems

1. For each of the graphs below, find a Hamilton cycle, or explain why a Hamilton cycle cannot exist.



2. a) Show that for all $n \geq 5$, the graph $\overline{C_n}$ (the complement of the n-vertex cycle graph) is Hamiltonian.

   b) Find a 5-vertex graph G such that neither G nor $\overline{G}$ is Hamiltonian.

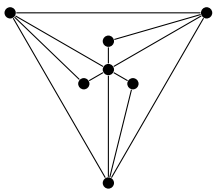
3. a) Find a Hamilton cycle in the cube graph Q_3.

   b) Prove that the hypercube graph Q_n has a Hamilton cycle for all $n \geq 3$, by induction on n.

   c) A Gray code is an alternate binary encoding of numbers with a useful property: to go from the encoding of k to the encoding of $k+1$, only one bit needs to be changed.

   Explain how to obtain a Gray code for the numbers $0, 1, \ldots, 2^n - 1$ using a Hamilton cycle in Q_n.

4. In Chvátal's original paper defining tough graphs [16], the following example is given:



   a) Prove that this graph is not Hamiltonian.

   b) Prove that this graph is tough.

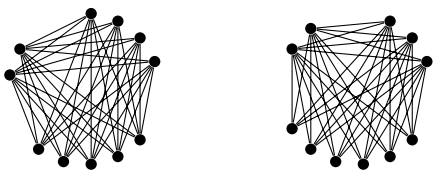
5. Let G be a bipartite graph with bipartition (A, B). Prove that if there is a subset $S \subseteq A$ with $|N(S)| < |S|$, then G cannot be Hamiltonian. (This statement is also a consequence of the statement of the next problem, but it can be proved directly.)

6. Prove that every tough graph with an even number of vertices (not necessarily bipartite) has a perfect matching.

7. Let G be an arbitrary graph, and let H be the graph obtained by adding a new vertex to G, adjacent to all vertices which already existed in G. Prove that G is traceable (G has a Hamilton path) if and only if H is Hamiltonian (H has a Hamilton cycle).

   (This argument is the reason why we do not spend much time thinking about traceable graphs and Hamilton paths: we can use it to take almost any fact about Hamilton cycles and turn it into a fact about Hamilton paths, instead.)

8. A complete tripartite graph is formed by taking three groups of vertices A, B, and C, then adding an edge between every pair of vertices in different groups: this is a generalization of complete graphs and complete bipartite graphs. We write $K_{a,b,c}$ for the complete tripartite graph with $|A| = a$, $|B| = b$, and $|C| = c$. For example, below are diagrams of $K_{2,4,5}$ (left) and $K_{2,3,6}$ (right).



a) One of these graphs is Hamiltonian. Find a Hamilton cycle in that graph.

b) The other graph is not Hamiltonian. Give a reason why it does not have a Hamilton cycle.

c) Generalize: find and prove a rule that tells you when $K_{a,b,c}$ is Hamiltonian. You may assume $a \leq b \leq c$.

9. a) Prove that all graphs with the degree sequence $7, 7, 7, 7, 7, 7, 5, 5, 2$ are Hamiltonian.

b) Find a general condition on degree sequences $d_1, d_2, \ldots, d_n$ which guarantees that all graphs with that degree sequence are Hamiltonian, and which generalizes your argument from part (a).

10. (BMO 2011) Consider the numbers $1, 2, \ldots, n$. Find, in terms of n, the largest integer t such that these numbers can be arranged in a row so that all consecutive terms differ by at least t.

18 Cliques and independent sets

The purpose of this chapter

The focus of this chapter betrays my interests: more than half of it has ended up related to Ramsey numbers and Ramsey's theorem. At the same time, I am laying the groundwork for several topics in the next chapter: interval graphs, greedy algorithms, and even random graphs with no large cliques or independent sets will all help us understand vertex coloring.

I briefly considered whether this book has too many chess puzzles in it, but the contrast between the queen placement problem in this chapter and the rook placement problem in Chapter 13 will be very useful for us as an example later on in Chapter 20. (I mention some of the connections in this chapter as well, so be sure you remember what the matching number $\alpha'(G)$ and vertex cover number $\beta(G)$ mean.)

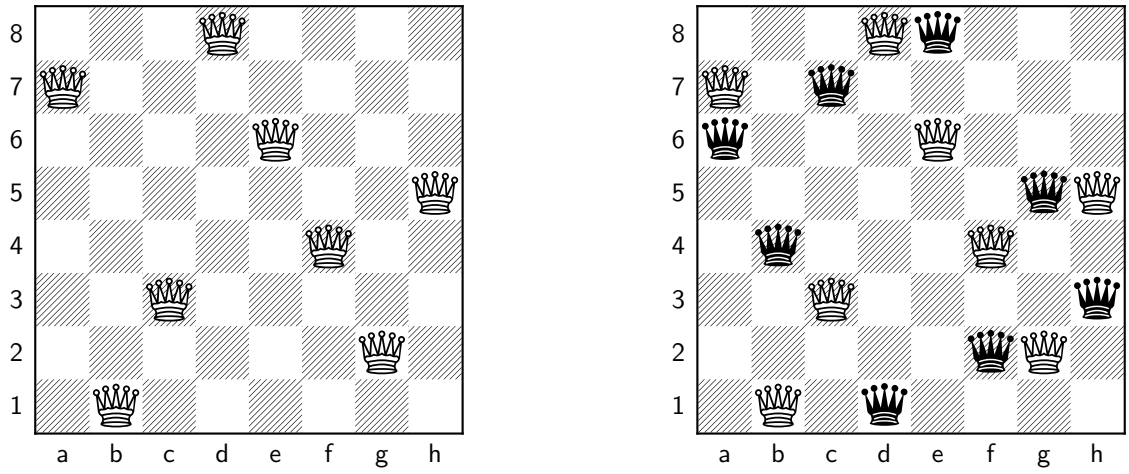
There is some redundancy built into my presentation of several of these topics, which explains how long this chapter has gotten. Proposition 18.2 and Theorem 18.4 arrive at almost the same destination, one using algorithms and the other by induction. Before the proof of Theorem 18.5, I present the same argument with concrete numbers. I do this so that you (as a reader or as a teacher) can choose which of these work best for you.

18.1 Two problems about queens

We have already looked at several puzzles about chess pieces, with rooks appearing in Chapter 13 and knights appearing in Chapter 17. Now it is time to look at the most powerful chess piece of all: the queen. The queen is a chess piece that can move any number of squares horizontally, vertically, or diagonally.

The classic 8 queens puzzle is the following: can you place 8 queens on a standard 8×8 chessboard so that none of the queens attack each other—that is, none of them can move to a square occupied by another? This is a strictly harder version of the problem posed in Chapter 13, where we placed rooks under the same condition, because a queen has all the movement potential of a rook, and more. It is still true that we cannot place two queens in the same rank or the same file, and so 8 queens is the absolute limit. However, putting all the queens in different ranks and files is not enough, due to the diagonal moves.

An early summary of the problem was given in *Mathematical Recreations and Essays* by W. W. Rouse Ball in 1892, republished many times since [6]. A total of 92 solutions are possible, but if we consider two solutions to be the same if they differ only by a rotation or reflection, then there are only 12 different possible solutions to consider. One of them is shown in Figure 18.1a; can you find a different solution yourself?



(a) 8 non-attacking queens

(b) 8 non-attacking queens, twice

Figure 18.1: Solutions to two puzzles about queen placements

We can keep going. Figure 18.1b shows two simultaneous solutions to the 8 queens puzzle that do not share any squares: one with 8 white queens, and one with 8 black queens. An even more impressive solution was found by Rob Pratt [85], who placed six solutions to the 8 queens puzzle on a chessboard simultaneously, and verified that more is impossible.

Question:	In fact, without seeing either of the diagrams in Figure 18.1, you can argue that if a solution to the 8 queens puzzle exists, then we can solve the version with 16 queens as well. How?
Answer:	Combine a solution to the 8 queens puzzle with its mirror image! The mirror image of a queen's location cannot be occupied by another queen, because then those two queens would attack each other.

Both of these problems can be modeled using graph theory. For the 8 queens puzzle itself, we define the 8×8 *queen graph* to have 64 vertices, one for each square of the chessboard, with an edge for between every two vertices that share a rank, file, or diagonal. Then our goal is to select 8 vertices from this graph with no edges between them.

For the problem of simultaneous solutions in multiple colors, define a graph whose 92 vertices are the 92 different possible solutions to the 8 queens puzzle, Make two vertices adjacent if the solutions overlap, so that they cannot be placed on the chessboard simultaneously. Then our goal is to select as many vertices as possible from this graph with no edges between them—another instance of the same problem!

In general, we make the following definitions:

Definition 18.1. *An **independent set** in a graph G is a set of vertices $I \subseteq V(G)$ such that no two vertices in I are adjacent. The **independence number** of G, denoted $\alpha(G)$, is the number of vertices in a maximum independent set in G.*

We defined the graphs in our previous problems such that edges represent conflicts, and so the set that we want to choose is a set of vertices with no edges between them. Instead, we could have defined the graphs so that edges represent compatibility. That is, two squares on the chessboard would be adjacent if two queens on those squares do not attack each other, and two solutions to the 8 queens puzzle would be adjacent if they do not overlap. This graph would be the complement of the 8×8 queen graph.

If we had done this, we would be looking for a different object, which has its own definition.

Definition 18.2. *A **clique** in a graph G is a set of vertices $Q \subseteq V(G)$ such that every pair of vertices in Q is adjacent. The **clique number** of G, denoted $\omega(G)$, is the number of vertices in a maximum clique in G.*

It is hard to say why α (alpha) and ω (omega) were chosen to represent these quantities, but at the very least there is some relationship there: α is the first letter in the Greek alphabet, and ω is the last.

Previously, we've referred to complete graphs as cliques, and this is not a coincidence: a set of vertices Q is a clique precisely when the induced subgraph $G[Q]$ is a copy of the complete graph $K_{|Q|}$. In principle, we could have defined cliques to be complete subgraphs of G, but this is not as useful. The edges in the subgraph don't tell us anything: they are simply all present.

Question:	What kind of subgraph is induced by a m-vertex independent set?
Answer:	A copy of $\overline{K_m}$, also known as the empty graph.

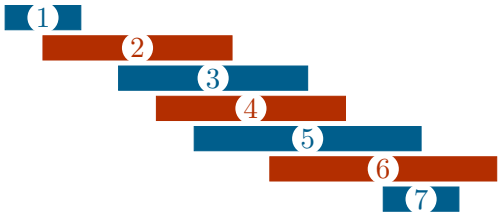
The queen graph is a bit asymmetric: the corner and edge vertices have fewer neighbors. A more mathematically elegant version of the problem makes all vertices equal, by allowing queen moves that wrap around the board. For example, the diagonal through the squares f1, g2, h3 would continue to a4, b5, and so on, as though we had glued opposite sides of the board to make a cylinder.

More generally, we define the periodic $n \times n$ queen graph to have vertices (x, y) where $1 \leq x \leq n$ and $1 \leq y \leq n$; here, x and y represent the horizontal and vertical coordinates of a square. Two vertices (x_1, y_1) and (x_2, y_2) are adjacent when $x_1 = x_2$ or $y_1 = y_2$ or $x_1 + y_1 \equiv x_2 + y_2 \pmod{n}$ or $x_1 - y_1 \equiv x_2 - y_2 \pmod{n}$. The "mod n" is what makes the board wrap around. With this modification, the side and corner squares are now just like every other square of the board.

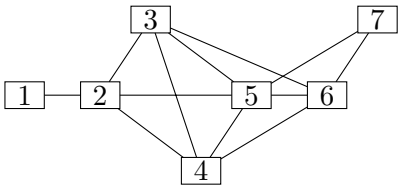
Unfortunately, there is no 8-vertex independent set in the periodic 8×8 queen graph: no solution to the 8 queens problem on a board that wraps around. George Pólya proved this in 1921 [84], along with a general statement: the periodic $n \times n$ queen graph has an n-vertex independent set exactly when n is not divisible by 2 or 3.

Remarkably, in such cases, we can even place n disjoint solutions on the $n \times n$ board, occupying every square! This object is known as a Knut Vik design, and has applications in minimizing the variability of statistical experiments. A Knut Vik design for all $n \times n$ boards where n is not divisible by 2 or 3 was first constructed by Samad Hedayat and Walter Federer in 1975 [55].

(In general, a partition of a graph into independent sets is called a *proper coloring*, and will be discussed in the next chapter.)



(a) 7 overlapping intervals



(b) The corresponding interval graph

Figure 18.2: Constructing an interval graph

18.2 Redundancies and relationships

Applications such as the 8 queens puzzle could be modeled either by cliques or by independent sets. Why, then, do we define both invariants to begin with? We could simply adopt the convention that we always use the "conflict" definition, where two vertices are adjacent if they're incompatible. Then, we'd always be looking for independent sets, and we'd never need to know the definition of a clique.

Question:	We can always reduce the problem of finding cliques in a graph G to finding independent sets, even if it did not arise from such an application. How?
Answer:	Take the complement graph $\overline{G}$, which has exactly the edges not present in G. Then a clique in G is precisely an independent set in $\overline{G}$.

One of the reasons we don't do this is that we might prefer one version of the graph for other reasons. This is true of the 8×8 queen graph where two vertices are adjacent if a queen on one square attacks the other, for example. In this graph, walks correspond to possible sequences of queen moves, which could have other applications. So if we want to keep the same definition of the graph for multiple puzzles, this forces our hand.

Sometimes it is even interesting to study both cliques and independent sets in the same graph. For example, this occurs in some scheduling problems. If we are scheduling a number of events (such as presentations at a conference or a convention) then we might want to think about which events are happening at the same time: we could make a graph where every event is a vertex, and two vertices are adjacent if the event times overlap.

Slightly more generally, suppose we have a set of intervals

$$[a_1, b_1], [a_2, b_2], \ldots, [a_n, b_n]$$

in the real line. (In the scheduling problem, a_i and b_i are the starting and ending time of the i^{th} event.) Then we can define a graph whose vertices are these intervals, with an edge whenever two intervals overlap. Such a graph is called an **interval graph**.[28]

Figure 18.2 shows a collection of intervals, together with the corresponding interval graph.

[28] Often, we also say that a graph isomorphic to a graph obtained in this way is also an interval graph. This way, being an interval graph becomes a true graph invariant, and we can discuss the structural question of which graphs can be represented as interval graphs.

Question:	In an interval graph representing a schedule of events, what would a clique represent?
Answer:	A clique corresponds to a set of intervals that all overlap: a set of events all happening at the same time. (You might care, for example, because you want to put them all in different rooms.)

Question:	What would an independent set represent?
Answer:	An independent set corresponds to a set of disjoint intervals: a set of events that don't conflict. (You might care, for example, because you want to attend all of them.)

Returning to the original topic, another reason to study both cliques and independent sets is that we might study the relationship between these objects and other properties of the graph. For example, we might ask: how does the maximum degree of a graph affect the clique number, and how does it affect the independence number? Both are valid questions, with very different answers.

In fact, there are already such connections to be explored. The independence number $\alpha(G)$ is closely related to both invariants introduced in Chapter 14: the matching number $\alpha'(G)$ and the vertex cover number $\beta(G)$.

As far as the matching number goes, you might be a bit suspicious due to the similar notation. It is not an accident: the $'$ symbol indicates that $\alpha'(G)$ is the "edge version" of $\alpha(G)$. Where $\alpha(G)$ is the maximum number of vertices that do not share any edge, $\alpha'(G)$ is the maximum number of edges that do not share any vertex. We will explore this connection and others like it in more detail in Chapter 20.

It is the connection to $\beta(G)$ that you might be surprised by.

Question:	Suppose U is a particularly small vertex cover in a graph G: a set of vertices including at least one endpoint of every edge. How can we use it to find a particularly large independent set in G?
Answer:	The set $I = V(G) - U$ consisting of all vertices not in U is an independent set![29] If two vertices $x, y \in I$ were adjacent, then U would not contain any endpoint of the edge xy, contradicting its status as a vertex cover.

Numerically: $\alpha(G) = |V(G)| - \beta(G)$. Once again, the question arises: why do we have two invariants $\alpha(G)$ and $\beta(G)$ when one of them can easily be used to obtain the other?

A good answer in this particular case is that we might be interested in ratios between the number of vertices in two independent sets, or two vertex covers. For example, if we have an algorithm that finds a vertex cover of at most $2\beta(G)$ vertices, or an independent set of at least

[29] Together, U and I make V, just as "you" and "I" make "we".

$\frac{1}{2}\alpha(G)$ vertices, we might say: that's within a factor of 2 of the true answer, so the guarantee is good enough.

In fact, such an approximation algorithm for vertex covers does exist—but such an approximation algorithm for independent sets does not, as far as we know! That's because the relationship $I = V(G) - U$ does not play well with ratios: doubling the size of the set U is not at all the same thing as halving the size of the set I.

18.3 Greedy algorithms

In general, algorithms that find maximum cliques or maximum independent sets must do so by backtracking: pick a vertex, try including it, and if it doesn't work out, try leaving it out. This is better than trying all $2^{|V(G)|}$ sets of vertices one by one: if we pick a vertex, then we can make some deductions about which other vertices cannot be picked. However, even the best backtracking algorithms take an exponentially long time to finish.

If we want an answer quickly, we must give up on insisting that we get the right answer. The most basic technique is to use a greedy algorithm: to pick vertices to add to our set without considering the consequences for future decisions, and to see where that gets us.

Here is one possible greedy algorithm for finding an independent set I in a graph G. We begin by setting $I = \varnothing$; we also keep track of a set A which contains all the vertices that are still available for use. Initially, $A = V(G)$. A single iteration of the greedy algorithm consists of the following steps:

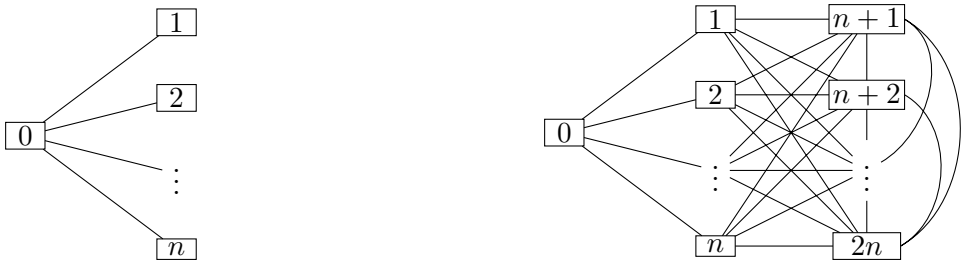
1. Let x be any vertex in A; add it to the independent set, replacing I by $I \cup \{x\}$.
2. Replace A by $A - (\{x\} \cup N(\{x\}))$: remove both x and $N(\{x\})$ (the set of vertices adjacent to x) from A.
3. If $A = \varnothing$, stop.

It is impossible for the set I to ever contain two adjacent vertices x and y: if we ever add x to I, we remove y from A, and if we ever add y to I, we remove x from A. Therefore the output of this algorithm is guaranteed to be an independent set.

Question:	If we want a greedy algorithm for finding a clique, how should we modify this algorithm?
Answer:	At each iteration, we should replace A by $A \cap N(\{x\})$ instead, so that all vertices we add in future iterations are adjacent to x.

What's more, the output I is guaranteed to be a *maximal* independent set (following the maximal/maximum distinction introduced in Chapter 13). The set I is not necessarily the largest independent set, but there is no bigger independent set I' that contains all of I. This is true because every vertex $y \in V(G)$ is removed from A in some iteration, for one of two reasons:

1. y is equal to x, the vertex added to I in that iteration.



(a) A star graph

(b) Extending the star graph

Figure 18.3: Graphs which confuse the greedy independent set algorithm

2. y is adjacent to x.

In both cases, $I \cup \{y\}$ is not a bigger independent set. In the first case, that's because $I \cup \{y\}$ is the same set as I. In the second case, that's because $I \cup \{y\}$ is not an independent set; it contains the edge xy.

This proves that I is a maximal independent set. However, I is not guaranteed to be a maximum independent set: there may well be other independent sets which are larger.

Question:	Can you find a graph in which the set I we find is much smaller than another independent set?
Answer:	One of many examples is the graph shown in Figure 18.3a: a copy of the star graph S_{n+1}. If the algorithm adds vertex 0 to I, it will immediately get $A = \varnothing$ and stop; however, $I = \{0\}$ is much smaller than the independent set $\{1, 2, 3, \ldots, n\}$.

We might say: okay, the problem is clear. We chose a vertex with very high degree and removed all its neighbors from A, causing the algorithm to stop very quickly. What if we try to be smarter about step 1 of the algorithm, so that at each iteration we choose the vertex x with the fewest neighbors in A?

This might sometimes be a useful heuristic, but it is not guaranteed to work. Consider instead the graph in Figure 18.3b; here, we've added vertices $n+1, n+2, \ldots, 2n$, adjacent to vertices $1, 2, \ldots, n$ and to each other.

Question:	What will the greedy algorithm do, given the graph in Figure 18.3b, if it chooses the vertex with the fewest neighbors in A at each step?
Answer:	The first vertex chosen will still be vertex 0, because it has degree n and all other vertices have degree at least $n+1$. Then, vertices $1, \ldots, n$ will be deleted, and the second vertex will be one of $n+1, \ldots, 2n$. Then, all remaining vertices will be deleted, and the algorithm will find a 2-vertex independent set.

Question:	What is the largest independent set in the graph in Figure 18.3b?
Answer:	The set $\{1, 2, \ldots, n\}$, which is $\frac{n}{2}$ times bigger than the set found by the greedy algorithm.

There is no rule for choosing vertices in the greedy algorithm that's always guaranteed to do well. However, at the very least, we can use the greedy algorithm to prove worst-case bounds.

Proposition 18.1. *If G is an n-vertex graph with maximum degree $\Delta(G)$, then $\alpha(G) \geq \frac{n}{\Delta(G)+1}$.*

Proof. To prove the lower bound, we find an independent set in G using the greedy algorithm, taking no special care in choosing the vertex x at each iteration.

Nevertheless, what we can guarantee is that at each iteration, at most $\Delta(G) + 1$ vertices are removed from A. We remove x and all neighbors of the chosen vertex x in A. There are at most $\Delta(G)$ such neighbors; there could be fewer if $\deg_G(x) < \Delta(G)$, or if not all neighbors of x are still in A, but there cannot be more.

The algorithm does not stop until A is empty, but since each iteration removes at most $\Delta(G)+1$ out of n vertices from A, there must be at least $\frac{n}{\Delta(G)+1}$ iterations. At each iteration, one vertex is added to I; therefore, the output of the algorithm is an independent set I with at least $\frac{n}{\Delta(G)+1}$ vertices. Therefore $\alpha(G)$ is also at least $\frac{n}{\Delta(G)+1}$. □

Question:	What lower bound can we get for the clique number $\omega(G)$?
Answer:	Since $\omega(G) = \alpha(\overline{G})$, we have $$\omega(G) \geq \frac{n}{\Delta(\overline{G}) + 1} = \frac{n}{n - \delta(G)},$$ where $\delta(G)$ is the minimum degree of G.

Here, the identity $\Delta(\overline{G}) = n - 1 - \delta(G)$ holds because a maximum-degree vertex in $\overline{G}$ is a minimum-degree vertex in G, and because $\deg_{\overline{G}}(x) = n - 1 - \deg_G(x)$ for any vertex x.

18.4 An indecisive algorithm

In preparation for our next topic, consider a variant of the greedy algorithm that hasn't made up its mind yet about whether it wants to find a clique or an independent set. It will still keep track of a set A of available vertices, initially equal to $V(G)$. It will also build up a set we'll simply call S, initially equal to $\varnothing$; at each iteration, it will move a vertex x from A to S, then update A.

The upside to being indecisive is that at each iteration, the algorithm has a choice:

- It could remove x and all of its neighbors from A.
- It could remove x and all vertices which are not its neighbors from A.

If the algorithm always chooses the option that keeps A as large as possible, then it's guaranteed to keep at least $\frac{|A|-1}{2}$ vertices for the next iteration (rounded up). It must always remove x, but it can keep at least half of the remaining vertices.

The downside to being indecisive is that the final set S will not be either a clique or an independent set. Suppose that after k iterations, we obtain $S = \{x_1, x_2, \ldots, x_k\}$, where x_i is the vertex added to S at the i^{th} iteration.

Question:	Is there any useful property at all that this set S has?
Answer:	We can at least guarantee that a vertex x_i is either adjacent to all of the vertices $x_{i+1}, x_{i+2}, \ldots, x_k$, or to none of them. The first case occurs when we removed all the non-neighbors of x_i from A in the i^{th} iteration; the second case occurs when we removed all the neighbors of x_i from A in the i^{th} iteration.

This property seems hard to make use of, but let's try. Suppose we call a vertex x_i a "Q-type" vertex (Q for clique) if it is adjacent to all of $x_{i+1}, x_{i+2}, \ldots, x_k$, and a "I-type" vertex (I for independent set) if it is adjacent to none of these vertices. We can write S as the union $Q \cup I$, where Q is the set of all Q-type vertices and I is the set of all I-type vertices.

Question:	What can we say about Q and I?
Answer:	Q is a clique and I is an independent set.

Question:	What is the type of vertex x_k?
Answer:	It is both Q-type and I-type, since there are no vertices that come after it, so both conditions are vacuously true.

Except for x_k, every vertex is only one of Q-type or I-type, so $|Q| + |I| = |S| + 1$. Therefore, if the indecisive algorithm only picks at the last possible moment whether to return the clique Q or the independent set I, it is guaranteed to find a set of $\frac{|S|+1}{2}$ vertices.

The indecisive algorithm is not usually useful in practice: for most problems, we have already decided whether we are looking for a clique or an independent set. However, we can use it to prove a relationship between these quantities.

Proposition 18.2. *If G is a graph with at least 2^{n-2} vertices, then $\alpha(G) + \omega(G) \geq n$. In other words, $\alpha(G) + \omega(G) \geq 2 + \log_2 |V(G)|$.*

Proof. Apply the indecisive algorithm to G. At the beginning, $|A| \geq 2^{n-2}$. After one iteration, we replace A by a set with at least $\frac{|A|-1}{2}$ elements in it, so we can say that $|A| \geq \frac{2^{n-2}-1}{2}$; rounding up, $|A| \geq 2^{n-3}$. Similarly, after a second iteration, we can still guarantee $|A| \geq 2^{n-4}$. This argument can be repeated for as long as our lower bound on $|A|$ is even, ensuring that $\frac{|A|-1}{2}$ can be rounded up to $\frac{|A|}{2}$.

After the $(n-2)^{\text{th}}$ iteration, we can guarantee that $|A| \geq 2^{n-(n-2)-2}$, or $|A| \geq 1$. This is still enough to know that there will be an $(n-1)^{\text{th}}$ iteration. However, it's possible that the $(n-1)^{\text{th}}$ iteration is the last, because it removes the only remaining element in A.

At the end, the indecisive algorithm chooses between returning a clique Q or an independent set I, which satisfy $|Q| + |I| = |S| + 1$. Since $|S| \geq n - 1$, we conclude that $|Q| + |I| \geq n$. This completes the proof, since $\alpha(G)$ is at least $|I|$ and $\omega(G)$ is at least $|Q|$: the largest clique and independent set are at least as large as the ones we found. □

Intuitively, we can make the independence number $\alpha(G)$ small by giving G lots of edges, and we can make the clique number $\omega(G)$ small by giving G few edges. However, these strategies directly oppose each other, so it makes sense that we wouldn't be able to keep both $\alpha(G)$ and $\omega(G)$ from growing. Proposition 18.2 is the first result we will prove that quantifies this relationship.

18.5 Ramsey numbers

We can improve on Proposition 18.2 by noticing a flaw in how the indecisive algorithm chooses which vertices to keep in A. All it does is pick the option that keeps A as large as possible, but if you want the final clique or independent set to be as large as possible, that's not always correct.

Question:	Why would you ever decide to keep a smaller set to be set A in the next iteration?
Answer:	Suppose that the current set S contains many Q-type vertices and few I-type vertices. Then you might keep all the neighbors of x in A, making x another Q-type vertex, even if the algorithm stops a bit sooner as a result.

To be strategic about it, we need to quantify the exact trade-off: how many fewer vertices in A are an acceptable price to pay in exchange for guaranteeing a vertex of the right type? This depends on our goals, of course. If we'd be happy with a clique of size 100 and we already have 98 Q-type vertices, we should make x another Q-type vertex at almost any cost—even if x only has one or two neighbors in A. If we're further from our goal, we should be more conservative.

That's still vague; to make it more concrete, we need to know exactly how many vertices in A are necessary to reach any particular goal we might have. So for integers $k \geq 1$ and $l \geq 1$, we define the *Ramsey number* $R(k, l)$ to be the least integer n such that any n-vertex graph G satisfies either $\alpha(G) \geq k$ or $\omega(G) \geq l$.

Question:	Can we be certain that any such integer n exists?
Answer:	Yes, by Proposition 18.2. Take a graph G with $n = 2^{k+l-3}$ vertices; then $\alpha(G)+\omega(G) \geq k+l-1$, which cannot be true if both $\alpha(G) \leq k-1$ and $\omega(G) \leq l-1$. Therefore 2^{k+l-3} vertices are enough, and $R(k, l)$ is at most this big.

Ramsey numbers are named after Frank Ramsey, who was the first to prove a result analogous to Proposition 18.2 in 1930 [89], though he was not interested in concrete bounds on the necessary number of vertices.

Let me describe a concrete situation to motivate our need for Ramsey numbers. Suppose we are looking for a clique or an independent set of size 5. Currently, we have 3 vertices in S: two Q-type vertices (adjacent to each other and all vertices in A) and one I-type vertex (with no neighbors in A). We've chosen a vertex x in the fourth iteration, and we're considering what to do with the neighbors of x to prepare for the next iteration.

1. If we keep all the neighbors of x in A, then x will be another Q-type vertex. We will need to find either two more Q-type vertices or four more I-type vertices in the future.
2. If we keep all the non-neighbors of x in A, then x will be another I-type vertex. We will need to find either three more Q-type vertices or three more I-type vertices in the future.

Question:	How should we choose between these two options?
Answer:	If x has at least $R(4,2)$ neighbors in A, then the first option is guaranteed to succeed. If x has at least $R(3,3)$ non-neighbors in A, then the second option is guaranteed to succeed. If neither is true, then both options are risky, and we might need to guess.

Some people are very comfortable with algorithms; some people prefer more mathematical arguments. In case you are the second type of person, you will be happier with the following lemma, whose proof is inspired by the indecisive algorithm but doesn't use it directly.

Lemma 18.3. *For all integers $k \geq 2$ and $l \geq 2$, the Ramsey number $R(k,l)$ is at most the sum $R(k,l-1) + R(k-1,l)$.*

Proof. Let G be an arbitrary graph with $R(k-1,l) + R(k,l-1)$ vertices. Our goal is to prove that this is enough to guarantee that either $\alpha(G) \geq k$ or $\omega(G) \geq l$.

Choose any vertex $x \in V(G)$, and divide $V(G) - \{x\}$ into two sets: A_1, the vertices adjacent to x, and A_2, the vertices not adjacent to x.

It is impossible that both $|A_1| < R(k,l-1)$ and that $|A_2| < R(k-1,l)$. In that case, $|V(G)| = |A_1|+|A_2|+1$ would be at most $(R(k,l-1)-1)+(R(k-1,l)-1)+1$ or $R(k-1,l)+R(k,l-1)-1$: one less than the number of vertices we defined G to have.

If $|A_1| \geq R(k,l-1)$, then the induced subgraph $G[A_1]$ is big enough to either contain a k-vertex independent set I or an $(l-1)$-vertex clique Q, by the definition of the Ramsey number $R(k,l-1)$. Therefore G contains either a k-vertex independent set I or an l-vertex clique $Q \cup \{x\}$ (since x is adjacent to all vertices in A_1, and in particular all vertices in Q).

If instead $|A_2| \geq R(k-1,l)$, then $G[A_2]$ is big enough to either contain a $(k-1)$-vertex independent set I or an l-vertex clique Q. Therefore G contains either a k-vertex independent set $I \cup \{x\}$ or an l-vertex clique Q.

In all cases, we either find a k-vertex independent set or and l-vertex clique in G. Since G was chosen to be an arbitrary graph with $R(k-1,l) + R(k,l-1)$ vertices, this proves that $R(k,l) \leq R(k-1,l) + R(k,l-1)$. □

Lemma 18.3 is the foundation of a recurrence that can be used to get upper bounds on the Ramsey numbers. To use the recurrence, we also need base cases.

Question:	Lemma 18.3 starts at $k \geq 2$ and $l \geq 2$. What is $R(k, l)$ when $k = 1$ or when $l = 1$?
Answer:	In all these cases, it is just 1; even something like $R(1, 10000)$ is 1. There is only one possible 1-vertex graph, and it has both a 1-vertex clique and a 1-vertex independent set.

Question:	$R(k, 2)$ and $R(2, l)$ can also be computed exactly; what are they?
Answer:	$R(k, 2) = k$ and $R(2, l) = l$. For the first of these, suppose we have a k-vertex graph. Either it has no edges (and there is a k-vertex independent set) or it has an edge xy (and $\{x, y\}$ is a 2-vertex clique). The argument for $R(2, l)$ is similar.

The general upper bound on $R(k, l)$ using this recurrence is sometimes also called Ramsey's theorem, but actually, it was only proven in 1955 by Robert Greenwood and Andrew Gleason [40] (who also proved Lemma 18.3 in the process).

Theorem 18.4. *For all integers $k \geq 1$ and $l \geq 1$, $R(k, l) \leq \binom{k+l-2}{k-1}$.*

Proof. We induct on $k + l$. (That is, if $P(n)$ is the statement that the theorem holds for all integers $k \geq 1$ and $l \geq 1$ with $k + l = n$, then we prove $P(n)$ for all $n \geq 1$ by induction on n.)

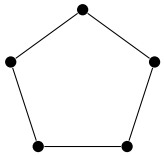
As our base case, we can take any of $k + l = 1$ (where there is nothing to prove), or $k + l = 2$ (where $R(1, 1) = 1 = \binom{0}{0}$), or $k + l = 3$ (where $R(2, 1) = 1 = \binom{1}{0}$ and $R(1, 2) = 1 = \binom{1}{1}$), but eventually we will have to induct.

Consider some $k \geq 1$ and $l \geq 1$ with $k + l \geq 3$, and assume the following induction hypothesis: for all $k' \geq 1$ and $l' \geq 1$ with $k' + l' = k + l - 1$, we have the upper bound $R(k', l') \leq \binom{k'+l'-2}{k'-1}$. Actually, we only need it for two values of k' and l': the ones that appear in the right-hand side of Lemma 18.3.

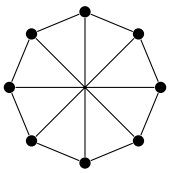
Question:	If $k = 1$ or $l = 1$, we cannot apply Lemma 18.3; what do we do?
Answer:	If $k = 1$ or $l = 1$, we know that $R(k, l) = 1$, and in both cases the upper bound $\binom{k+l-2}{k-1}$ simplifies to $\binom{l-1}{0} = 1$ or $\binom{k-1}{k-1} = 1$, so we do not even need the induction hypothesis.

Otherwise, $k \geq 2$ and $l \geq 2$. By Lemma 18.3, $R(k, l) \leq R(k, l-1) + R(k-1, l)$, and both of the Ramsey numbers on the right-hand side are suitable for the induction hypothesis. We can conclude that

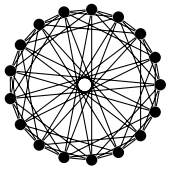
$$R(k, l) \leq \binom{k + (l-1) - 2}{k-1} + \binom{(k-1) + l - 2}{(k-1) - 1} = \binom{k+l-3}{k-1} + \binom{k+l-3}{k-2}.$$



(a) The cycle graph C_5



(b) $\mathrm{Ci}_8(1,4)$



(c) $\mathrm{Ci}_{17}(1,2,4,8)$

Figure 18.4: Lower bounds for a few small Ramsey numbers

The right-hand side of this inequality simplifies directly to $\binom{k+l-2}{k-1}$. This is called Pascal's identity, and if you haven't seen it before, there is a short combinatorial argument. By its combinatorial definition, $\binom{k+l-2}{k-1}$ counts the $(k-1)$-element subsets $S \subseteq \{1, 2, \ldots, k+l-2\}$. These come in two types. If $k+l-2 \notin S$, then S is a subset of $\{1, 2, \ldots, k+l-3\}$, and so there are $\binom{k+l-3}{k-1}$ cases of this type. If $k+l-2 \notin S$, then $S - \{k+l-2\}$ is a $(k-2)$-element subset of $\{1, 2, \ldots, k+l-3\}$, and so there are $\binom{k+l-3}{k-2}$ cases of this type.

Altogether, there must be $\binom{k+l-3}{k-1} + \binom{k+l-3}{k-2}$ subsets. We already know there are $\binom{k+l-2}{k-1}$ subsets, so these two expressions must be equal. Returning to our upper bound on $R(k,l)$, we can replace one expression by the other and get $R(k,l) \leq \binom{k+l-2}{k-1}$. This proves the induction step, completing the proof of the theorem. □

Both Ramsey and later Greenwood and Gleason also considered a generalized problem. Here, we define the Ramsey number $R(k_1, k_2, \ldots, k_m)$ to be the least n with the following property: whenever we write the complete graph K_n as the union of n-vertex graphs $G_1 \cup G_2 \cup \cdots \cup G_m$, there will be some G_i such that $\omega(G_i) \geq k_i$. Sometimes, this is phrased as coloring the edges of K_n using m different colors, but I will avoid this so that you don't confuse it with all the other ways we will color a graph.

18.6 Lower bounds

Once we have defined Ramsey numbers, it is tempting to try to figure out an exact formula for them, or at least an approximate one. Our first upper bound on $R(k,l)$ was 2^{k+l-3}, and our second bound of $\binom{k+l-2}{k-1}$ has a similar rate of growth, at least when $k = l$. Is this anywhere close to the truth? How would we know?

In the definition of the Ramsey number $R(k,l)$, we ask for the least n such that every n-vertex graph satisfies a condition. To prove that $R(k,l) \geq n$, it is enough to find a single $(n-1)$-vertex graph which does not yet satisfy the condition. (Such graphs are sometimes called Ramsey graphs, because of their use in proving lower bounds for Ramsey numbers.)

Figure 18.4 shows a few of the graphs we need.

- The cycle graph C_5, shown in Figure 18.4a, has no 3-vertex clique or 3-vertex independent set. This proves that $R(3,3) \geq 6$: we need at least 6 vertices to guarantee one of these objects, because 5 are not enough.

  In fact, Theorem 18.4 proves an upper bound of $\binom{3+3-2}{3-1} = \binom{4}{2} = 6$ on $R(3,3)$, so we know that $R(3,3) = 6$.

- The circulant graph $\text{Ci}_8(1,4)$ shown in Figure 18.4b has no 3-vertex clique or 4-vertex independent set, and it has 8 vertices, so it proves that $R(4,3) \geq 9$. Theorem 18.4 only proves that $R(4,3) \leq \binom{4+3-2}{4-1} = \binom{5}{3} = 10$, but in one of the practice problems, I will ask you to improve this upper bound: in fact, $R(4,3) = 9$.

Question:	What about $R(3,4)$?
Answer:	It is also equal to 9; in fact, $R(k,l)$ and $R(l,k)$ are always equal. The lower bound that proves $R(3,4) \geq 9$ is the complement of $\text{Ci}_8(1,4)$: the circulant graph $\text{Ci}_8(2,3)$. A similar trick of complements can be used to show that every n-vertex graph contains a k-vertex independent set or an l-vertex clique, then it is also true that every n-vertex graph contains an l-vertex independent set or a k-vertex clique.

- Taking $R(3,4) = R(4,3) = 9$ for granted, it follows from Lemma 18.3 that $R(4,4) \leq R(4,3) + R(3,4) = 18$. In fact, $R(4,4) = 18$; to prove this, we need a 17-vertex graph.

  The graph in Figure 18.4c is one such graph: the circulant graph $\text{Ci}_{17}(1,2,4,8)$. This graph has no 4-vertex clique or 4-vertex independent set, so it proves the lower bound $R(4,4) \geq 18$.

Of course, we cannot go on forever like this; to give a general lower bound, some general scheme for constructing these graphs is needed. Unfortunately, it has been very difficult to construct these graphs. We run into two kinds of problems. First, most simple rules defining an arbitrarily large graph will give it a large clique or a large independent set. Second, when we eventually do find a general construction that looks promising, we often find out that proving something about its cliques or independent sets turns into an unsolved problem in number theory.

(Why number theory? Well, the offsets of the circulant graph $\text{Ci}_{17}(1,2,4,8)$ in Figure 18.4c are not arbitrary: they are the quadratic residues modulo 17. That is, every perfect square is congruent modulo 17 to one of ± 1, ± 2, ± 4, or ± 8. Graphs constructed in this way are called Paley graphs, and they are often useful for proving lower bounds on Ramsey numbers; however, we don't understand quadratic residues well enough to say what the lower bounds are in general.)

The best we've been able to do is to give up and pick a graph at random, showing that the result is very unlikely to have a large clique or a large independent set. Let me first illustrate this with concrete numbers. Suppose we define a graph G on 1000000 vertices by flipping a coin for every possible edge to see if it's present or absent. What are the odds that this random graph G will have a 40-vertex clique?

There are $\binom{1000000}{40} \approx 1.22 \times 10^{192}$ ways to choose 40 of the vertices of G. Each one of those 40-vertex sets could in theory be a 40-vertex clique... but that's very unlikely. There are $\binom{40}{2} = 780$ edges between those 40 vertices, so we flip 780 coins to decide which edges between the vertices are present. In order for us to get a clique, all the coin flips have to go one way, which has a probability of $2^{-780} \approx 1.57 \times 10^{-235}$.

If you bought $\binom{1000000}{40}$ tickets for a lottery that chose one winning ticket out of 2^{780}, then to find your total chances of winning, it would be enough to multiply these two numbers together. This would give us $(1.22 \times 10^{192}) \cdot (1.57 \times 10^{-235})$ or about 1.93×10^{-43}. The clique problem

is actually even worse, due to overlaps. After we consider the first clique, the second doesn't contribute its whole 2^{-780} probability, because some of those cases were already counted: the two cliques appear simultaneously!

Rather than consider what the situation with overlaps is, we can simply say that the probability of having a 40-vertex clique is at most 1.93×10^{-43}. We get the same probability for a 40-vertex independent set: once again, all the coin flips have to go one way for all 780 edges between vertices of a set. Since both probabilities are tiny, we conclude that this random 1000000-vertex graph is almost certain not to have a 40-vertex clique or 40-vertex independent set, proving that $R(40, 40) > 1000000$.

In 1947, Pál Erdős proved [29] that this gives an exponential lower bound on the Ramsey numbers:

Theorem 18.5. *For all $k \geq 3$, $R(k, k) > 2^{k/2}$.*

Proof. Pick a random graph on $n = 2^{k/2}$ vertices by flipping a coin for every edge. There are $\binom{n}{k}$ possible k-vertex sets; each has a $2^{-\binom{k}{2}}$ chance of being a clique, and another $2^{-\binom{k}{2}}$ chance of being an independent set, for a total probability of $2^{1-\binom{k}{2}}$ of doing anything of interest.

As before, we get an upper bound on the probability that any of these cliques or independent sets are created by multiplying these quantities together: $\binom{n}{k} \cdot 2^{1-\binom{k}{2}}$. It is always true that $\binom{n}{k} \leq \frac{n^k}{k!}$, and for $k \geq 3$, it is true that $k! > 2^k$. Applying both inequalities, our upper bound becomes $n^k \cdot 2^{-k} \cdot 2^{1-k^2/2+k/2}$ or $2^{1-k/2}$, which is less than 1 as soon as $k \geq 3$.

Therefore the probability that the random graph doesn't do what we want is less than 1: some of the n-vertex graphs we could get have neither k-vertex cliques nor k-vertex independent sets. This shows that $R(k, k) > n$. □

The bounds $R(k, k) > 2^{k/2}$ and $R(k, k) \leq \binom{2k-2}{k-1}$ (which grows roughly as 2^{2k}) are fairly far apart, though they are both exponential; for example, they tell us that $32 < R(10, 10) \leq 48620$. It has been at least 70 years since any of the results mentioned in this chapter so far; has there been improvement since then?

There has been and there still is a lot of work done on specific small Ramsey numbers. A dynamic survey of the best-known bounds is maintained by Stanisław Radziszowski [88]. For reference, at the time of writing, it gives the bounds $798 \leq R(10, 10) \leq 16064$: much better than 32 and 48620.

Improvements to general bounds on $R(k, k)$ have also been made. When I first taught graph theory, I had to conclude this topic by saying that the bases of the exponents in the bounds ($\sqrt{2}$ for the lower bound and 4 in the upper bound) are still untouched, with improvements only in polynomial factors. As of 2023, that is no longer true: a paper by Campos, Griffiths, Morris, and Sahasrabudhe [12] followed by another from Gupta, Ndiaye, Norin, and Wei [43] have brought down the upper bound to roughly $R(k, k) \leq 3.8^k$.

18.7 Practice problems

1. Consider the graph whose vertices are the ordered pairs (x, y) where $1 \leq x \leq a$ and $1 \leq y \leq b$, and where two vertices (x, y) and (x', y') are adjacent whenever $x \neq x'$. (This is a complete a-partite graph where each part has size b.)

   a) What is the clique number of this graph? Describe what the largest cliques look like.

   b) What is the independence number of this graph? Describe what the largest independent sets look like.

2. Let G be the graph with vertex set $\{2, 3, \ldots, 100\}$ in which two vertices are adjacent if one is divisible by the other.

   a) What is the independent set in G found by a greedy algorithm which looks at all the vertices of G in ascending order? (And why is this a uniquely bad order in which to look at the vertices?)

   b) Find the independence number of G.

   c) Find the clique number of G.

3. Take an $n \times n$ grid and number the rows and columns 1 through n. Then fill the grid as follows: in the entry in row i and column j, write the remainder when $i + 2j$ is divided by n: a number between 0 and $n - 1$. (This is the construction found by Hedayat and Federer in [55].)

   Prove that that when n is not divisible by 2 or 3, these labels form a Knut Vik design: if we pick any label between 0 and $n - 1$ and place a queen on all entries with that label, then this is a set of n non-attacking queens, even if diagonals wrap around.

4. a) Verify that the graphs in Figure 18.4 really do prove the lower bounds $R(3, 3) \geq 6$, $R(4, 3) \geq 9$, and $R(4, 4) \geq 18$, by checking that they have no cliques or independent sets of the relevant sizes.

   b) Prove that $R(4, 3) \leq 9$ by contradiction, supposing there is a 9-vertex graph with no vertex x that has $R(4, 2)$ neighbors or $R(3, 3)$ non-neighbors. What would the degree sequence of this graph be?

   If you prefer, prove the generalization: when $R(k, l - 1)$ and $R(k - 1, l)$ are both even, then Lemma 18.3 can be improved to $R(k, l) \leq R(k, l - 1) + R(k - 1, l) - 1$.

5. In terms of n, find the minimum and maximum number of edges in a $3n$-vertex graph with independence number $2n$.

6. In this problem, you will see one possible deterministic alternative to Theorem 18.5, and see how it compares to the random construction.

   Let G_1 be the 5-cycle C_5. Then, for each $k > 1$, construct G_k by starting with G_{k-1}, and then replacing

   - Each vertex x of G_{k-1} by five vertices x_1, x_2, x_3, x_4, x_5 with a 5-cycle through them;
   - Each edge xy of G_{k-1} by 25 edges $x_i y_j$ for $1 \leq i, j \leq 5$.

In other words, we replace the vertices of G_{k-1} by copies of C_5.

a) Prove that $\alpha(G_2) = \omega(G_2) = 4$.

b) Prove that $\alpha(G_k) = \omega(G_k) = 2^k$. (Induct on k.)

c) We get a lower bound $R(17, 17) > n$ by finding an n-vertex graph G with $\alpha(G) \leq 16$ and $\omega(G) \leq 16$. What lower bound does the construction in this problem give for $R(17, 17)$, and how does it compare to the lower bound from Theorem 18.5?

What if we try to bound $R(33, 33)$ instead?

7. An open problem called Conway's 99-graph problem, already mentioned in Chapter 4, is to determine whether there is a 99-vertex graph with the following properties:

- Every two adjacent vertices have exactly one common neighbor;
- Every two non-adjacent vertices have exactly two common neighbors.

If such a graph exists, it is known (spoilers for the previous practice problem) that it is 14-regular.

Suppose G is a 99-vertex graph satisfying Conway's conditions. What is the best upper bound you can prove on $\alpha(G)$?

(When I tried solving this problem, I found three arguments, giving upper bounds of 43, 33, and 22. See how well you can do!)

8. (APMO 2003) Given two positive integers m and n, find the smallest positive integer k such that among any k people, either there are $2m$ of them who form m pairs of mutually acquainted people or there are $2n$ of them forming n pairs of mutually unacquainted people.

In other words, find the smallest k such that for any k-vertex graph G, either $\alpha'(G) \geq m$ or $\alpha'(\overline{G}) \geq n$, where $\overline{G}$ is the complement of G.

9. This question is more of a puzzle than a mathematical problem. If 8 queens are placed on the chessboard as in Figure 18.1a, there is a closed knight's tour of just the 56 unoccupied squares. Can you find it?

(Up to symmetry, this is the only solution to the 8 queens puzzle for which such a tour exists. Can you guess what goes wrong with the other solutions?)

19 Graph coloring

The purpose of this chapter

Graph coloring is one of the iconic problems in graph theory, though usually people think of it in the context of coloring maps. This application was mentioned in Chapter 1, and we will see it in detail later, in Chapter 24. The applications you will see in this chapter are more metaphorical versions of coloring—but that's par for the course in graph theory, which is all about representing the world with metaphorical points and metaphorical lines.

I begin this chapter with one upper bound and two lower bounds on chromatic number, and I think these are the basics you should know of the theory of graph coloring. Meanwhile, Theorem 19.4 and Theorem 19.7 demonstrate how these basics can be used. The motivation for these specific demonstrations is to show you an example where the clique bound of Proposition 19.2 is very useful, and an example where it's not at all useful; in either case, I hope that it gives you a sense of why the clique number does not necessarily tell us the chromatic number.

Brooks' theorem (which describes the conditions under which $\chi(G) \leq \Delta(G)$) is a common result to cover in an introductory graph theory class, but I've left it out. I have done this because I think seeing it for the first time at this point is extraordinarily uncompelling—and I say this as someone who has both used the theorem in research and taught a week-long class about variants of the theorem. When you just start out learning about graph coloring, it is unsatisfying to go to great effort to improve Proposition 19.1 by a tiny amount. I recommend seeing Brooks' theorem for the first time as a more experienced graph theorist, for example by reading "Brooks' Theorem and Beyond" by Cranston and Rabern [21], which presents a variety of proofs of the theorem and its extensions.

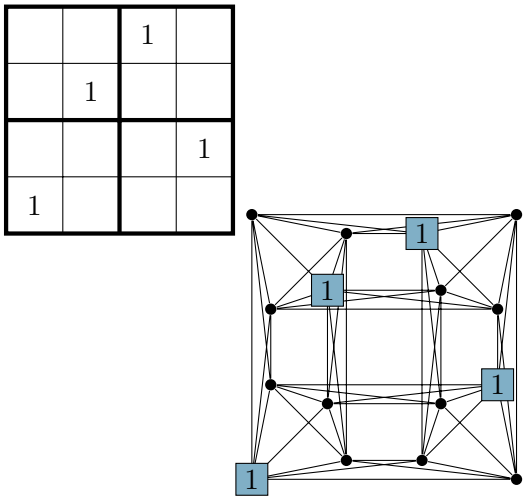
19.1 Graph coloring and Sudoku

In case you have never encountered a Sudoku puzzle, let me explain how they work. Figure 19.1a shows an example of a Sudoku puzzle. It is a 9×9 grid, with its 81 cells grouped together into 9 boxes. Some of the cells already have numbers in them. The objective of the puzzle is to figure out which numbers should go in the empty cells! The rule is that in each row, each column, and each 3×3 box, the nine cells should contain the numbers 1 through 9 in some order.

For example, the top left corner of Figure 19.1a could not contain the numbers $2, 3, 6, 8$, because they are already present in its 3×3 box; it could also not contain the numbers $1, 5$ (already present in its row) or 9 (already present in its column). The numbers 4 or 7 have not been ruled out, though only one of them is correct; the other would eventually lead to a contradiction if you wrote it in, even though there's no obvious problem yet.

	6	3				5	1	
2			3					8
8			7					4
				4		2	9	
			1	3	5			
	7	6		2				
9					8			2
6					2			1
	2	4				6	8	

(a) A Sudoku puzzle



(b) The 4×4 Sudoku graph

Figure 19.1: Sudoku puzzles and graph coloring

As usual, we would like to turn this logic puzzle into a problem of graph theory. The first task is to choose a graph to model it with. Although other options exist, for us it will be most convenient to make the 81 cells in the grid be the vertices of the Sudoku graph. Make two cells adjacent in this graph if they are in the same row, the same column, or the same 3×3 box.

Question: How can the rules of Sudoku be expressed in terms of the edges of this graph?

Answer: The goal of Sudoku is now to assign one of the numbers 1 through 9 to each vertex of the graph, so that adjacent vertices are assigned different numbers.

Since a diagram of this Sudoku graph would look like utter chaos and not give you any intuition at all, I have drawn a diagram of the 4×4 Sudoku graph in Figure 19.1b. In 4×4 Sudoku, the rules are the same, except that each row, column, and 2×2 box has four numbers in it, and the cells are allowed to contain the numbers 1 through 4.

Question: In Figure 19.1b, the 1's are all entered into the grid (and shown in the graph). In general, what can you say about the set of all vertices assigned the number 1 in a Sudoku puzzle?

Answer: Two adjacent vertices cannot both be assigned the number 1, so the set of vertices with a 1 must be an independent set.

The graph-theoretic rules of Sudoku happen to match a well-studied problem in graph theory, which is why I used Sudoku as an example to begin this chapter. For historical reasons, rather than assigning every vertex a number, we assign every vertex a color. For practical reasons, graph theorists are very flexible about what "colors" are. It's fine to say, "We will use the set of colors $\{1, 2, \ldots, 9\}$," for example, in which case the numbers in a Sudoku grid are our colors.

Definition 19.1. *A **proper coloring** (or just **coloring**) of a graph G is a function $f\colon V(G) \to C$ such that for all edges $xy \in E(G)$, $f(x) \neq f(y)$. The elements of C are called colors, but can be any set we like.*

Allow me to justify my terminology. The term "proper coloring" is the formal name for a coloring f with the condition that $f(x) \neq f(y)$ for all edges xy, so that's what we should be using. However, almost every single time we color something in this book, we will want a proper coloring. To omit needless words, I will simply say "coloring" unless emphasis is needed: for example, when we want to check that a coloring f really is proper. On the rare occasion where we want a coloring that might give the same color to both endpoints of an edge, I will call it an *improper coloring* to emphasize this.

Back to graph theory! Sometimes it is convenient to think about coloring a graph in a different way: not as a function, but as a partition. We say that a **color class** of a coloring is a set of of all vertices of some color. Every vertex is in exactly one color class, so the color classes are a partition of $V(G)$; a coloring is proper if and only if the endpoints of every edge are in different color classes.

Question:	What are the color classes in the solution to a Sudoku puzzle?
Answer:	There are 9 of them: for each number from 1 to 9, the set of all cells containing that number is a color class.

The study of graph coloring originated with a very practical coloring problem: given a map divided into several regions, how can we color the regions so that neighboring regions receive different colors? This is a coloring of the *graph of adjacencies* of the map: its vertices are the regions, with an edge between every pair of regions that share a border. We will not investigate this problem in detail in this chapter; we will return to it in Chapter 24, once we know more properties of graphs we can obtain from a map.

One thing is still missing from our definition, however. It is too easy to color a graph: just give every vertex its own color! We are not very happy with this solution, for the same reason that Sudoku puzzles would not be very fun if you could solve them by writing the numbers $1, 2, \ldots, 81$ in the squares in any order. We are more interested in a coloring of a graph if it uses fewer colors; here are some definitions to express this.

Definition 19.2. *A k-coloring of G is a coloring of G in which the set of colors has size k. A graph which has a (proper) k-coloring is called **k-colorable**.*

*The **chromatic number** $\chi(G)$ is the least value of k for which G is k-colorable.*

Question:	The Sudoku graph is 9-colorable; is it 8-colorable?
Answer:	No: all 9 vertices in a row (or column, or box) must use different colors, so at least 9 vertices are necessary. This is because the vertices in a row, column, or box form a clique.

Finding the chromatic number $\chi(G)$ is an optimization problem, like many we've seen before, and there are several observations we can make which are common to all optimization problems. (See Appendix A for more details.) Determining that $\chi(G) = k$ always comes down to two steps: we must show that $\chi(G) \leq k$, and that $\chi(G) \geq k$. These look like similar inequalities, but they're very different in practice.

- Every coloring of a graph G gives us an upper bound on $\chi(G)$: a k-coloring of G tells us that $\chi(G) \leq k$. A single example is a proof; it might be hard to find, but it is short.
- To prove a lower bound $\chi(G) \leq k$, we must show that no $(k-1)$-colorings of G are possible. Unless we think of something clever, we must resort to an exhaustive search.

We will see arguments for both lower and upper bounds in this chapter; however, the graph coloring problem is a hard one in general, so we will not always be able to make the bounds meet.

Question: For the clique number $\omega(G)$ and the independence number $\alpha(G)$, this is reversed: lower bounds are proved by example, and upper bounds need an exhaustive argument. Why?

Answer: Both $\omega(G)$ and $\alpha(G)$ are maximization problems: we are trying to find the largest clique or independent set. When coloring a graph, we are trying to use the smallest set of colors: $\chi(G)$ is a minimization problem.

That being said, the special case of 2-coloring is much easier than the general case, and we have already encountered it in this book.

Question: What name do we already have for 2-colorable graphs?

Answer: They are precisely the bipartite graphs! The two color classes in a 2-coloring are precisely the two sides of a bipartition.

Intuitively, finding a 2-coloring is easier because, as soon as you give a vertex a color, you eliminate that color as an option for its neighbors, leaving only one option for them. This breaks down if a third color is available: if you give a vertex a color, its neighbors still have two options left.

19.2 Greedy coloring

Just now, I told you that a single example of a k-coloring is enough to prove that a graph G is k-colorable. While that's true, it's often not good enough from a theoretical point of view, because we want to prove general results. To deal with a whole family of graphs and not just one graph, we need a strategy for coloring them. From a practical point of view, of course, a strategy for coloring graphs is also very important.

(a) A bad vertex ordering (b) A good vertex ordering

Figure 19.2: Two ways to order the vertices of a tree before coloring it greedily

The simplest graph coloring algorithm is the greedy coloring algorithm. It could be briefly summarized as, "Go through the vertices and give each vertex the first color not used on its neighbors." To make the algorithm well-specified, though, so that we know what it does in every situation and can use it in proofs, we need to be explicit about the choices we're making.

First of all, we should give the colors an order, so that the phrase "the first color not used" makes sense. It does not matter how we do this, because the set of colors does not matter beyond its size. We can say that the set of colors is $\{1, 2, \ldots, k\}$ for some k, in which case the first unused color is just the least unused number. Often, we don't know how many colors we'll need when we start color, so we can begin by allowing all the positive integers, and then narrow down the set of colors to the ones we ended up using.

More importantly, we should give the vertices an order: the order in which we color them. This choice is vitally important to the outcome. Consider the two diagrams in Figure 19.2, which show the same tree (or isomorphic trees). Of course, we know that the tree is 2-colorable, because all trees are bipartite—but the greedy algorithm doesn't know!

Question:	What happens if you greedily color the vertices of the tree by going left to right in Figure 19.2a?
Answer:	The colors used will be $1, 2, 1, 3, 1, 2, 1, 4$ in that order; we use 4 colors.

Question:	What happens if you greedily color the vertices of the tree by going left to right in Figure 19.2b?
Answer:	The colors used will be $1, 2, 1, 2, 1, 2, 1, 2$ in that order; we use 2 colors.

In the practice problems at the end of this chapter, I will ask a few more questions about these examples. To summarize, if the vertices are ordered badly, then the greedy algorithm can color even a tree with arbitrarily many colors. However, if you know an optimal way to color a graph ahead of time, you can always convince the greedy algorithm to do just as well. Finding a good way to order the vertices without knowing the answer in advance is the hard part.

Even if we don't have a good idea for how to order the vertices, though, there are some worst-case guarantees.

Proposition 19.1. *Any graph G with maximum degree $\Delta(G)$ is $(\Delta(G) + 1)$-colorable.*

Proof. Give the vertex set $V(G)$ an arbitrary order $x_1, x_2, \ldots, x_n$, and color G using the greedy algorithm, using the positive integers as the set of colors.

For each i, when the greedy algorithm gets to vertex x_i, that vertex has at most $\Delta(G)$ neighbors among $x_1, x_2, \ldots, x_{i-1}$. At most $\Delta(G)$ different colors can be represented on those neighbors. Therefore, at least one of the colors $1, 2, \ldots, \Delta(G)+1$ does not appear on any of x_i's neighbors.

The greedy algorithm will never use a number higher than $\Delta(G)+1$ if a lower one is available, so it will give x_i some color from the set $\{1, 2, \ldots, \Delta(G)+1\}$. This is true for each i, so at the end, the greedy algorithm has found a coloring using at most $\Delta(G)+1$ colors. □

The bound of Proposition 19.1 is better than having no bound at all, but it's not very good. However, that does not mean that the greedy algorithm is no good! In this chapter and later in the book, we will see examples where we can deduce better upper bounds from the greedy algorithm for specific families of graphs. To do this, we will use the structure of those graphs to come up with a better ordering of the vertices.

(Of course, there are also proofs based on more sophisticated algorithms.)

19.3 Two lower bounds

If we want to know the chromatic number of any graph for certain, it's not enough to find a coloring: we must also prove that we used the least number of colors possible. To do this, we need lower bounds.

Unfortunately, a typical answer to why a graph is not 7-colorable (for example) might be, "We tried all possible ways to color the vertices, by making guesses and backtracking and trying something else when they were wrong, and never arrived at a 7-coloring of the entire graph." State-of-the-art techniques might combine this with a sequence of partial deductions of a standard type, but these are outside the scope of this textbook and can still be exponentially long in some cases.

In order to make something resembling progress, we will need to lower our standards considerably. First, we will accept lower bounds that are often far from the truth, in the hope that sometimes they will still help us. Second, we will accept lower bounds that are themselves difficult to find, as long as they provide us with a short proof of a lower bound. This is more useful in theoretical problems, where we can generalize that proof to deal with a family of graphs at once.

To arrive at the first such lower bound, we begin by asking: what's the most straightforward graph that requires k colors? It is the complete graph K_k: its k vertices are all adjacent, so all of them need different colors.

Moreover, if a large graph G contains a copy of K_k inside it, then we know that $\chi(G) \geq k$ as well. Even the copy of K_k inside G needs k colors: coloring the other vertices can only make things worse. (In general, if H is a subgraph of G, then $\chi(G) \geq \chi(H)$, because to color G, we must first color H.)

Copies of K_k inside G correspond to k-vertex cliques. The size of the largest clique in G is the clique number $\omega(G)$. We conclude:

Proposition 19.2. *For any graph G, $\chi(G) \geq \omega(G)$.*

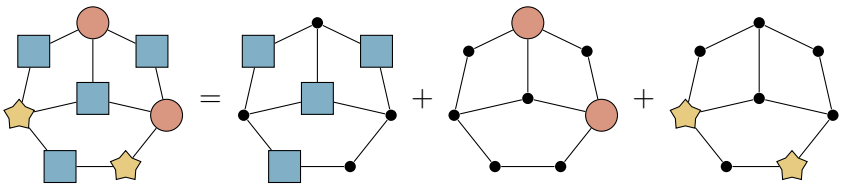


Figure 19.3: A 3-coloring of a graph viewed as a partition into color classes

To be honest, I suspect that I don't need to work hard to prove to you that Proposition 19.2 is true. The difficult thing for most people to accept—and we will see later in this book that this has been difficult for serious mathematicians as well as beginners in graph theory—is that Proposition 19.2 does not tell us everything: that $\chi(G)$ is not always equal to $\omega(G)$.

To see why this is true, we can start by understanding when $\chi(G) \geq 3$, because another way to say "$\chi(G) \geq 3$" is "G is not bipartite", and we already know a lot about bipartite graphs.

Question:	What does Proposition 19.2 say about when a graph is not bipartite?
Answer:	It says that $\chi(G) \geq 3$ whenever $\omega(G) \geq 3$: that G is not bipartite whenever G contains a copy of K_3.

Question:	What do we already know about subgraphs that mean a graph is not bipartite?
Answer:	By Theorem 13.1, such subgraphs are copies of $C_3, C_5, C_7, C_9, \ldots$: cycles of odd length.

Of these graphs, C_3 is isomorphic to K_3: in both cases, we have three vertices, and any two are adjacent. So we might say that Proposition 19.2 identifies the smallest fundamental non-bipartite graph, but misses all the rest.

For a higher number of colors, the reality is even more complicated, and Proposition 19.2 is even more of an oversimplification. Still, it is a useful lower bound that is often correct in practice.

What else can we use?

The independence number $\alpha(G)$ is the exact opposite of the clique number $\omega(G)$: it is the size of the largest set of vertices with no edges between them. So it may seem surprising that the independence number can also help us put lower bounds on the chromatic number.

To see the connection, we switch to thinking of a coloring as a partition into color classes; Figure 19.3 shows an example of such a partition. We started with the definition that a coloring is proper when the endpoints of any edge are in different color classes; however, another way to say this is that no two vertices in the same color class are adjacent. In other words, a coloring is proper if and only if every color class is an independent set.

The independence number $\alpha(G)$ tells us how big such color classes can get, and we can use this to deduce another lower bound on the chromatic number of G.

Proposition 19.3. *For any n-vertex graph G, $\chi(G) \geq \frac{n}{\alpha(G)}$.*

Proof. The independence number $\alpha(G)$ is the largest number of vertices in any independent set, so in particular every color class in a k-coloring has at most $\alpha(G)$ vertices. Together, the k color classes contain at most $k \cdot \alpha(G)$ vertices. Every vertex must be in one of the color classes, so if there are n vertices, then $k \cdot \alpha(G) \geq n$. Rearranging, we get $k \geq \frac{n}{\alpha(G)}$. □

If Proposition 19.3 and not Proposition 19.2 is the reason why the chromatic number of a graph is large, then this could be a very subtle reason. We might not notice, when coloring a small portion of the graph, that there is a problem; the effect of having no large independent sets is only noticeable when we look at the graph as a whole.

But is this scenario actually possible? Are there graphs G where the independence number, and not the clique number, controls $\chi(G)$?

In fact, not only do such graphs exist, but they are very common. In Chapter 18, we explored what happens when a graph on 1000000 vertices is constructed by flipping coins to for each possible edge to decide if we include it. We discovered that it is extremely unlikely for this graph to have cliques or independent sets with more than 40 vertices.

Question:	What would Proposition 19.2 tell us about such a graph?
Answer:	If there are no cliques with more than 40 vertices, the best we could hope is a lower bound of $\chi(G) \geq 40$, and we might not even be able to get that.

Question:	What would Proposition 19.3 tell us about such a graph?
Answer:	If there are 1000000 vertices, but there are no independent sets of more than 40 vertices, then at least $\frac{1000000}{40} = 25000$ colors are needed in a coloring.

By flipping a coin for every edge, we make every 1000000-vertex graph equally likely. This means that, if something is true with a very high probability for this random graph, it is true for almost all 1000000-vertex graphs. In other words, $\chi(G)$ is much larger than $\omega(G)$ for almost all of these graphs!

This does not mean, however, that Proposition 19.2 is useless. While random graphs are a good source of examples, they do not actually reflect most graphs we study: if a graph is interesting to us, there is a good chance that it is atypical somehow. This example does not tell the whole story.

In the rest of this chapter, we will see some alternative scenarios: a class of graphs in which the lower bound of Proposition 19.2 always gives the chromatic number exactly, and class of graphs in which the chromatic number is much larger than both of our lower bounds.

(a) A coloring of an interval graph

(b) Solving the scheduling problem

Figure 19.4: Coloring an interval graph

19.4 Coloring interval graphs

In Chapter 18, we introduced interval graphs: graphs obtained from a collection of intervals, with an edge between vertices whenever the intervals overlap. These are a nice application of graph coloring for two reasons: first, coloring interval graphs has useful practical applications, and second, it is easier than the general graph coloring problem.

First, on the applications. Suppose that the intervals we use to build an interval graph are the start and end times of events: this is the setting we looked at in Chapter 18. We previously said that both cliques and independent sets have meaning in this setting: cliques are sets of events all happening at the same time, and independent sets are sets of events that don't conflict with each other.

Question:	Actually, there's a subtlety: by definition, a clique is a set of events where every pair has an overlap, but does this mean that there's a single point in time when all events in the clique overlap?
Answer:	Yes; to see this, take a clique of events, let a be the first of these events to end, and let z be the of these events to start. Events a and z must overlap, because they're part of a clique together. At any time in that overlap, all events in the clique have started (because even z has started) but none have ended (because even a hasn't ended).

The chromatic number of an interval graph is also a useful quantity. Suppose we want to pick locations for the events. Two events happening at the same time should be in different locations. At the same time, we would like to use as few locations as possible: this makes it easier to go from one event to another, and we might also have a limited number of locations available. Choosing locations for the events is exactly a graph coloring problem, where the locations are the colors; Figure 19.4 shows a coloring of an interval graph, and the corresponding solution to the scheduling problem for the intervals.

The same model can be useful in situations where the scheduling is more metaphorical. For example, graph coloring can be used in the register allocation problem in computer science [14]. Here, we want to assign the variables used by a program to a limited supply of memory locations called registers (if possible); however, two variables cannot be assigned to the same register if they are in use at the same time. In simple settings, the graph that represent conflicts between

the variables becomes an interval graph, and the assignment to registers is a coloring of that interval graph.[30]

The interesting thing about coloring interval graphs is that the complexity I alluded to in previous sections is almost not present. The greedy algorithm is enough to color them optimally, and the lower bound of Proposition 19.2 gives the exact chromatic number, with no funny business.

Theorem 19.4. *If G is an interval graph, then $\chi(G) = \omega(G)$. Moreover, if the greedy algorithm is given the vertices of G in order of the starting points of the intervals, then it will use only $\omega(G)$ colors.*

Proof. The second half of the statement of the theorem is actually all we need to prove. If the greedy algorithm uses only $\omega(G)$ colors, then $\chi(G) \leq \omega(G)$; however, by Proposition 19.2, $\chi(G) \geq \omega(G)$, so the two must be equal.

Let's consider an iteration of the greedy algorithm in which it must color a vertex x associated to interval $[a, b]$. To see which colors the algorithm cannot use, we look at the neighbors of x that already have a color. What are these neighbors? They correspond to intervals that have an overlap with $[a, b]$, but have starting points less than a.

In order to overlap $[a, b]$, these intervals must contain the point a. This means that they all overlap each other, too! In fact, x and the neighbors of x already given a color must form a clique: call this clique Q.

There are $|Q| - 1$ neighbors of x that have already been colored, because the last element of Q is x itself. Therefore at most $|Q| - 1$ colors are ruled out for x, and the greedy algorithm will be able to give x one of the first $|Q|$ colors on its list.

The clique Q can be bigger or smaller depending on x, but in all cases, $|Q| \leq \omega(G)$ by definition of the clique number. Therefore the greedy algorithm always uses one of the first $\omega(G)$ colors on the list, completing the proof. □

19.5 Mycielski graphs

We have already seen that in a randomly chosen graph, the chromatic number can be much higher than the clique number. But the truth is even worse than this: it is possible to increase the chromatic number arbitrarily high while the clique number does not grow at all, and find graphs where both Proposition 19.2 and Proposition 19.3 drastically underestimate the chromatic number!

One method for doing this is a construction found in 1955 by Jan Mycielski [76]. We will first see it as an operation on graphs: a graph that transforms a graph G into its Mycielskian $M(G)$, increasing its chromatic number but not its clique number. Then, the sequence of Mycielski graphs is obtained by applying this operation over and over again.

The *Mycielskian* $M(G)$ of a graph G is constructed in three steps:

[30] More sophisticated models can produce graphs with fewer edges than the interval graph, with a smaller chromatic number that's harder to compute, but I will not get into those details.

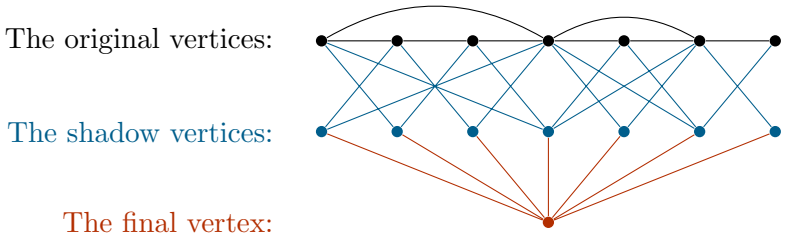


Figure 19.5: The Mycielskian of a graph

1. Start with a copy of G. (I will call the vertices in this copy of G the *original vertices*, for ease of reference.)
2. For each original vertex x, add a new vertex x' adjacent to the same original vertices as x. (I will call x' a *shadow vertex*: the shadow of x.) No edges are added between the shadow vertices.
3. Finally, add one more vertex (which I will call the *final vertex*) adjacent to all the shadow vertices.

Figure 19.5 illustrates this construction in an example. I should note that the terms "original vertices", "shadow vertices", and "final vertex" are made up by me; there's no standard terminology.

The first objective of this definition is to avoid increasing the clique number: we want $\omega(M(G))$ to stay equal to $\omega(G)$. Technically, this is violated when G has no edges: then $\omega(G) = 1$, but $M(G)$ will gain some edges when the final vertex is added, so $\omega(M(G)) = 2$. However, it will be true in all other cases.

Lemma 19.5. *For all graphs G with $\omega(G) \geq 2$, $\omega(M(G)) = \omega(G)$.*

Proof. We do not know anything about cliques formed by the original vertices: these are cliques that already existed in G. In particular, because $M(G)$ contains a copy of G, $\omega(M(G))$ is at least $\omega(G)$: every clique in G can also be found in $M(G)$. It will also be possible that $M(G)$ has new large cliques—but we will show that none of these are bigger than cliques G already had.

To see this, suppose that Q is a clique in $M(G)$. If $|Q| \leq 2$, then we know that $|Q| \leq \omega(G)$ because we assumed $\omega(G) \geq 2$, so we may assume that Q contains at least 3 vertices.

Question:	Can Q contain the final vertex?
Answer:	No, because none of the neighbors of the final vertex are adjacent to each other.

Question:	Can Q contain any shadow vertices?
Answer:	Yes, but at most one: no two shadow vertices are adjacent, so C cannot contain two or more shadow vertices.

If Q contains no shadow vertices at all, we are not interested: it's contained in a copy of G, therefore $|Q| \leq \omega(G)$. So suppose Q contains the original vertices $x_1, x_2, \ldots, x_{k-1}$ and the shadow vertex x'_k: the shadow of x_k. Since the original vertex x_k is not adjacent to its shadow x'_k, we know that $x_k \notin Q$.

By the construction of x'_k, among the original vertices, it has only the neighbors that x_k does: since x'_k is adjacent to $x_1, x_2, \ldots, x_{k-1}$, we know x_k must be adjacent to them, as well. Therefore $\{x_1, x_2, \ldots, x_k\}$ is also a clique, and it has the same number of vertices as Q.

However, $\{x_1, x_2, \ldots, x_k\}$ consists solely of original vertices, so it can have at most $\omega(G)$ vertices! This means that $|Q| \leq \omega(G)$ as well.

We've now established the inequality $|Q| \leq \omega(G)$ for all cliques Q in $M(G)$, so we conclude that $\omega(M(G)) \leq \omega(G)$. This concludes the proof. □

The most interesting case of Lemma 19.5 is when $\omega(G) = 2$. (Such graphs are sometimes called triangle-free graphs.) In this case, the graphs $M(G)$, $M(M(G))$, and so on all have clique number 2 as well: the least clique number that a graph with edges can have.

Meanwhile, the chromatic number of G ticks up!

Lemma 19.6. *For all graphs G, $\chi(M(G)) = \chi(G) + 1$.*

Proof. Here, we are expressing the solution to one optimization problem (the chromatic number of $M(G)$) in terms of the solution to another optimization problem (the chromatic number of G), so it's worth thinking carefully about what we need to prove.

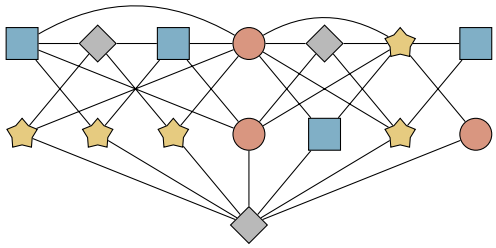
To show that $\chi(M(G)) \leq \chi(G) + 1$, we need to show that $M(G)$ has a coloring with $\chi(G) + 1$ colors. But we don't need to come up with the coloring of $M(G)$ from scratch: by definition of the chromatic number, we know that G has a $\chi(G)$-coloring, and we can use such a coloring to help us.

We might imagine that to show that $\chi(M(G)) \geq \chi(G) + 1$, we need to show that $M(G)$ cannot be colored with fewer colors, but proofs of impossibility are tricky: the paradigm of turning one coloring into another is much friendlier. So we rewrite this inequality as $\chi(G) \leq \chi(M(G)) - 1$, and now we're once again proving an upper bound on a chromatic number. We need to show that G has a coloring with $\chi(M(G)) - 1$ colors, and in the process, we may use a $\chi(M(G))$-coloring of $M(G)$.

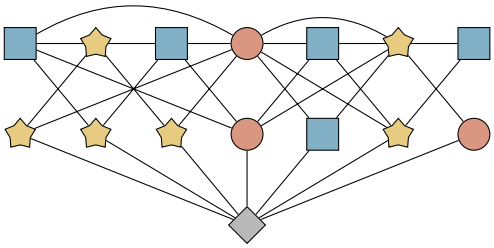
With the strategy talk out of the way, we can go on to the proof.

For the first step... I don't like phrases like, "The obvious thing will work," because what's obvious and what isn't depends on your experience. But the obvious thing will work. It's okay if it doesn't feel obvious yet; use it to train your intuition so that future arguments like it will be be more intuitive.

We start with a coloring of G using $\chi(G)$ colors. We would like to color $M(G)$ using only one more color than that. Since $M(G)$ starts with a copy of G inside it, the natural thing to do is to give every original vertex the same color as in the $\chi(G)$-coloring of G. Next, for each original vertex x, give its shadow x' the same color as x: this is guaranteed to work, because the vertices adjacent to x' are all original vertices adjacent to x, so none of them can have the same color as x. For the final vertex, use a new color we haven't used yet.



(a) A proper coloring of $M(G)$



(b) An improper coloring of $M(G)$

Figure 19.6: Going from a k-coloring of $M(G)$ to a $(k-1)$-coloring of G

The second step is harder. Let $k = \chi(M(G))$, so that there is a k-coloring of $M(G)$. We would like a coloring of G using only $k-1$ colors. Inside our coloring of $M(G)$, there is a coloring of $G\ldots$ but it may well use all k colors. Figure 19.6a shows an example, using the same $M(G)$ as in Figure 19.5.

We want to modify this coloring so that it only uses $k-1$ colors on the original vertices. Motivated by our first step, we would like the color used on the final vertex (let c be this color) to be the color we get rid of. It's a problem if any of the original vertices have color c; to "fix" it, whenever an original vertex x is assigned color c, we give x the color of its shadow x'.

Question:	What if the shadow x' is also assigned color c?
Answer:	That's impossible: x' is adjacent to the final vertex, which has color c, so x' must have some other color.

An example of the modified coloring we get is shown in Figure 19.6, and you can see from this example that it might be an improper coloring. In Figure 19.6, an original vertex recolored to ★ is adjacent to two shadow vertices that also have color ★. In general, when we give x the color of x', there is no guarantee that no neighbor of x has this color.

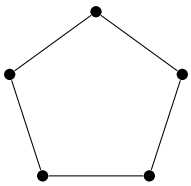
If we've gotten an improper coloring, surely our attempt was a failure? Not quite: remember, we only want a coloring of G, not all of $M(G)$. It turns out that if we only look at the original vertices, the coloring we have is a proper coloring!

To see this, let x and y be two adjacent original vertices. If neither x nor y had color c, then the colors of x and y are unchanged from the k-coloring we started with, so the colors of x and y are still different. It's also not possible that both x and y both had color c; because we started with a proper k-coloring.

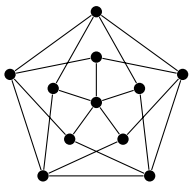
The last case is that only one of x and y (without loss of generality, x) had color c. In this case, the color of x is changed to the color of x' in the k-coloring. By the construction of $M(G)$, x' and y are also adjacent; therefore the color of x' is not the same as the color of y. It follows that x and y still have different colors after the change.

As a result, if we keep only the original vertices, then we do get a proper $(k-1)$-coloring of G. Therefore $\chi(G) \leq k-1$; since $k = \chi(M(G))$, this completes the second step and concludes the proof of the theorem. □

(a) M_2: the base case



(b) M_3: the 5-cycle



(c) M_4: the Grötzsch graph

Figure 19.7: The first few graphs in the sequence of Mycielski graphs

Lemma 19.5 and Lemma 19.6 are the two properties of the Mycielskian we needed to prove the following theorem.

Theorem 19.7. *For all $k \geq 1$, there exists a M_k with $\omega(M_k) \leq 2$ and $\chi(M_k) = k$.*

Proof. We induct on k. When $k = 1$ and when $k = 2$, a complete graph (K_1 and K_2, respectively) will do, so that's what we use for M_1 and M_2.

For the induction step, assume that we've already constructed a graph M_{k-1} with $\omega(M_{k-1}) \leq 2$ and $\chi(M_{k-1}) = k - 1$. Let $M_k = M(M_{k-1})$: the Mycielskian of M_{k-1}. By Lemma 19.5, $\omega(M_k) \leq 2$; by Lemma 19.6, $\chi(M_k) = \chi(M_{k-1}) + 1 = k$. These are exactly the two facts we needed to complete the induction step.

By induction, M_k exists for all $k \geq 1$. □

The graphs in the sequence $M_2, M_3, M_4, \ldots$ are called the *Mycielski graphs*. (We rarely include M_1, because it is not truly part of the recurrence: M_2 is not the Mycielskian of M_1, but a second base case.) Figure 19.7 shows the first few graphs in the sequence.

The graph M_3 is isomorphic to the cycle graph C_5, though it takes some "untwisting" to obtain a symmetric diagram from a 3-layer diagram in the style of Figure 19.5.

Finally, let help you make sense of how we get from C_5 to the diagram in Figure 19.7c. The original vertices have stayed in their place, with their shadows placed halfway to the center of the diagram; the final vertex placed at the center. The resulting graph, M_4, is known as the Grötzsch graph. It is named after Herbert Grötzsch, who used it in 1959 for the same purpose as Mycielski: as an example of a triangle-free graph with chromatic number 4 [42].

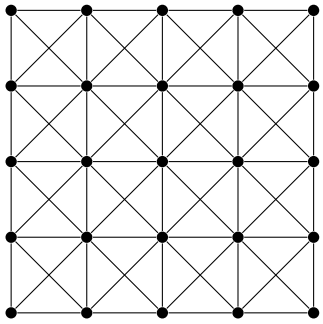
Question: Since the Mycielski graphs have clique number 2, Proposition 19.2 does not give a good lower bound on their chromatic number. What about Proposition 19.3?

Answer:

It is almost no better: the shadow vertices in M_k form an independent set containing almost half of all the vertices. Therefore Proposition 19.3 only proves that $\chi(M_k) \geq 3$.

19.6 Practice problems

1. Let G be the 25-vertex graph shown below:



   a) What is the clique number $\omega(G)$? What lower bound on $\chi(G)$ does it give?

   b) What is the independence number $\alpha(G)$? What lower bound on $\chi(G)$ does it give?

   c) What is the maximum degree $\Delta(G)$? What upper bound on $\chi(G)$ does it give?

   d) What is the actual chromatic number of G?

2. Find a greedy coloring of the circulant graph $\text{Ci}_9(1, 3)$, going through its verticesin numerical order. Is there a coloring of $\text{Ci}_9(1, 3)$ that uses fewer colors than the greedy coloring you found?

3. Let G_n be the graph with $V(G_n) = \{1, 2, \ldots, 3n\}$ in which vertices x and y are adjacent if $|x - y|$ is odd and not equal to 1.

   a) If G_n is colored greedily by going through the vertices in numerical order, how many colors are used? Find the answer in terms of n.

   b) What is the chromatic number of G_n, and why?

4. Alice and Bob take turns coloring the cube graph Q_3, one vertex at a time. They have a fixed set of colors $C = \{\text{red}, \text{blue}, \text{yellow}\}$, and neither player is allowed to give a vertex a color that has already been used on a neighbor of that vertex. However, they have different goals: Alice (who goes first) wants to get a coloring of Q_3, while Bob wants to arrive at a partial coloring which cannot be finished.

   If both players play as well as possible, which player will win?

5. Look back at Figure 19.2a, then:

   a) Find a tree and a vertex ordering of that tree for which the greedy algorithm will use 5 different colors.

   b) Prove that for all positive integers k, there is a tree and a vertex ordering of that tree for which the greedy algorithm will use k different colors.

6. The lowest-degree-last vertex ordering of an n-vertex graph G is constructed as follows, recursively. We choose the last vertex, x_n, to be any vertex whose degree is the lowest degree in G. To find the ordering $x_1, x_2, \ldots, x_{n-1}$ of the other vertices, we find the lowest-degree-last ordering of the $(n-1)$-vertex graph $G - x_n$.

   Prove that if G is a tree, then the greedy coloring algorithm, using the lowest-degree-last ordering, will never use more than 2 colors.

7. Let G be a graph with chromatic number k, and let $f\colon V(G) \to \{1, 2, \ldots, k\}$ be a k-coloring of G.

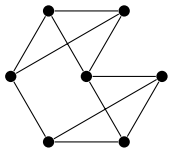
   Prove that if we sort the vertices of G by their f-values (putting all vertices given color 1 first and all the vertices given color k last), then the greedy algorithm will also find a k-coloring of G.

   (Be careful: if your proof shows that the greedy algorithm will find the coloring f we started with, then it's wrong, because that claim is false in general.)

8. a) Suppose that the greedy algorithm uses k colors on a graph G. Prove that G must have at least $1 + 2 + \cdots + (k-1) = \binom{k}{2}$ edges.

   b) Prove that if G has m edges, then $\chi(G) \leq 1 + \sqrt{2m}$.

9. The graph below is called the Moser spindle. It is a unit-distance graph: its vertices can be placed in the plane in such a way that two vertices are adjacent exactly when they are at distance 1 from each other.



   a) Find a unit-distance drawing of the Moser spindle.

   b) Prove that the Moser spindle has chromatic number 4.

   The chromatic number of the plane is the least number of colors needed to color every point of the plane so that no two points at distance from each other are the same color. Leo and William Moser constructed the Moser spindle in 1961 [75], proving that the chromatic number of the plane is at least 4.

   This was the best known lower bound until 2018, when Aubrey de Grey constructed a 1581-vertex unit distance graph with chromatic number 5 [41].

10. The wheel graph with n spokes is the graph obtained from the cycle graph C_n by adding a new vertex $n+1$ adjacent to all n vertices of the cycle.

    a) Draw a diagram of a wheel graph with 5 spokes.

    b) Show that this graph has clique number 3 but chromatic number 4.

    c) Generalize this construction to find a graph G in which $\omega(G) = k$ but $\chi(G) = k+1$, for every $k \geq 3$.

11. It is interesting to think about coloring the co-bipartite graphs: graphs whose complement is bipartite. A graph G is a co-bipartite graph if and only if we can write $V(G)$ as the disjoint union of two sets A and B which are both cliques in G (and, additionally, there may be some edges between A and B).

    a) Show that in any coloring of a co-bipartite graph, each color class contains at most 2 vertices.

    b) Suppose that a co-bipartite graph G has a coloring where k of the color classes have 2 vertices.

    How else can you describe those k color classes, and their role in the complement graph $\overline{G}$ (a bipartite graph)?

    c) What is the connection between a clique in a co-bipartite graph G and a vertex cover in its complement $\overline{G}$?

    d) Use parts (a)–(c) to prove that if G is a co-bipartite graph, then $\chi(G) = \omega(G)$.

12. The following inequality is true whenever $n_1, n_2, \ldots, n_k$ are positive integers with $n_1 + n_2 + \cdots + n_k = n$:

$$\sum_{i=1}^{k} \binom{n_i}{2} \geq k\binom{n/k}{2} = \frac{n(n-k)}{2k}.$$

    Use this inequality to find the maximum number of edges in an n-vertex graph with chromatic number k.

20 Line graphs

The purpose of this chapter

It is a little strange of me to put the chapter on line graphs in the part of the textbook devoted to hard problems, because there is not really a hard problem associated to line graphs. The opposite is true: the hard problems in this section are often easier to solve when we are solving them in a line graph. And that's exactly why I think these chapters belong together!

By looking at line graphs, we will also see many connections between problems that seemed very different before. There is an occasionally useful relationship between Euler tours (Chapter 8) in a graph G and Hamilton cycles (Chapter 17) in its line graph $L(G)$. Matchings (Chapter 13 through Chapter 16) are exactly the same thing as independent sets (Chapter 18) in the line graph. And by extending graph coloring to line graphs, we will arrive at edge coloring, a new problem with its own applications.

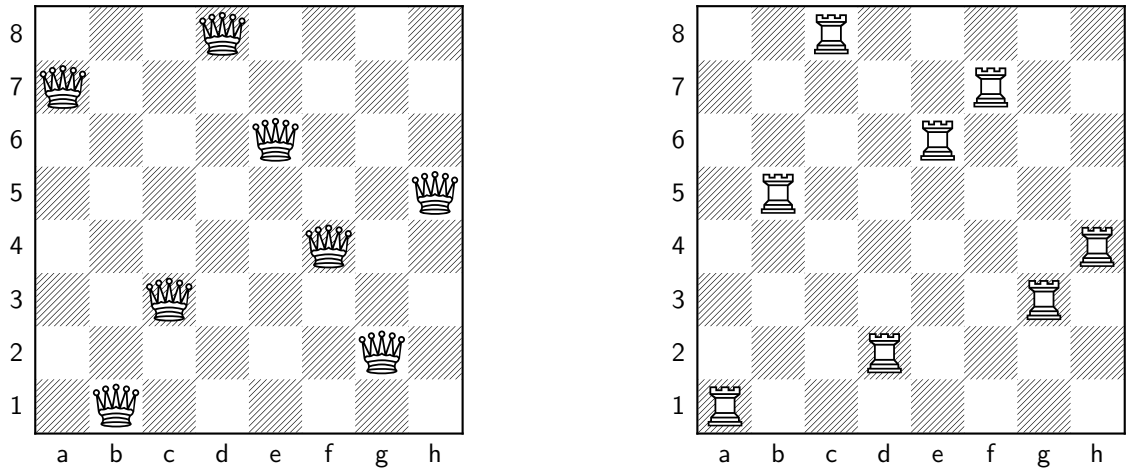
You can see from this list that to understand all the material in this chapter, you need to be familiar with many concepts in previous parts of the textbook. Apart from the chapters mentioned above, you should be familiar with Chapter 7, because both directed graphs and multigraphs will make an appearance.

It is not necessarily the case that you want to read the entire chapter if you're learning graph theory for the first time (or to teach the entire chapter in an introductory course). The application to de Bruijn sequences is interesting, but self-contained. Vizing's theorem is a natural observation to make after observing enough examples, but difficult to prove, though I have tried to pick out a less complicated proof.

20.1 Line graphs

In Chapter 13 and Chapter 18, I used two very similar problems as an example: the problem of placing eight rooks on a chessboard, and the problem of placing eight queens on a chessboard, so that they do not attack each other. The solutions are shown once again in Figure 20.1, for your reference. (Of course, the eight queens problem is strictly harder than the eight rooks problem, because a queen has all the movement power of a rook and more, so instead of Figure 20.1b, we could have just taken Figure 20.1a and changed all the queens to rooks.)

However, there's something strange about how we modeled these problems. The eight queens problem was our first example of finding independent sets in a graph: a hard problem we have no good algorithms for. It would have been equally natural to try solving the eight rooks problem in the same way. Define the 8×8 *rook graph* to be the graph whose 64 vertices are the squares of a chessboard, with an edge between two squares which are in the same rank or file. Then a solution to the eight rooks problem is an independent set in the 8×8 rook graph.



(a) Eight queens: an independent set

(b) Eight rooks: a... matching?

Figure 20.1: Rook and queen placements

We did something different in Chapter 13—and a good thing, too! Instead of finding an independent set, we were able to solve the problem by finding a matching, which is much easier. But why is it that the eight rooks problem has these two different models, and what is the relationship between them?

The answer is: the 8×8 rook graph happens to be a line graph, whose special structure makes many problems easier.

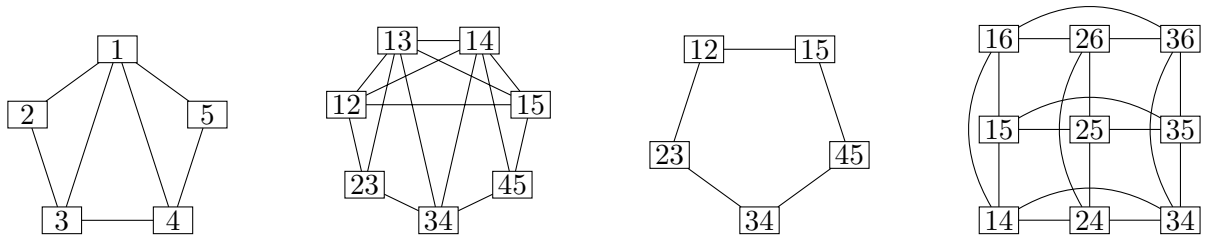
Definition 20.1. *The **line graph** $L(G)$ of a graph G is the graph whose vertices are the edges of G; two vertices e, e' of $L(G)$ are adjacent if and only if edges e and e' share an endpoint in G.*

Why is this the "line" graph? The name was introduced by Frank Harary and Robert Norman in 1960 [50], who referred to the vertices and edges of graphs as "points" and "lines". As a result, Harary and Norman called $L(G)$ the line graph because its vertex set (or "point set") was the set of "lines" of G. The name stuck.

Let's look at some examples. Figure 20.2a and Figure 20.2b show a graph G with nothing particularly special about it, and its line graph $L(G)$. The line graph has 7 vertices, which correspond to the 7 edges of G. Observe what happens to the four edges 12, 13, 14, and 15 incident to vertex 5: they turn into a 4-vertex clique in $L(G)$.

Most graphs have line graphs that are bigger than they are in every way, but not all. For example, Figure 20.2c shows the line graph of C_5: we see that $L(C_5)$ is isomorphic to C_5 itself, and the same will happen for every cycle graph.

Question:	What is the line graph of a path graph P_n isomorphic to?
Answer:	$L(P_n)$ is isomorphic to P_{n-1}: an n-vertex path has $n-1$ edges, each sharing an endpoint with the previous edge and the next edge along the path.



(a) G (b) $L(G)$ (c) $L(C_5)$ (d) $L(K_{3,3})$

Figure 20.2: Several examples of line graphs

Finally, Figure 20.2d shows the line graph of the complete bipartite graph $K_{3,3}$ (with vertices $\{1, 2, 3\}$ on one side and $\{4, 5, 6\}$ on the other side of the bipartition). It is also isomorphic to the 3×3 rook graph, showing the possible moves a rook could make on a 3×3 chessboard. With a little imagination, this should let you see how the 8×8 rook graph is isomorphic to the line graph $L(K_{8,8})$.

It is also possible to define the line digraph of a directed graph; the idea is similar, but two arcs that share an endpoint will only correspond to an arc in the line digraph when their orientations are compatible.

Definition 20.2. *The **line digraph** $L(D)$ of a directed graph D is the directed graph whose vertices are the arcs of D. For every pair of arcs (x, y) and (y, z) in D such that the first arc ends where the second arc starts, there is an arc $\big((x, y), (y, z)\big)$ in $L(D)$.*

It often turns out that one problem in graph theory is equivalent to another problem in disguise: solving one problem in the graph G is equivalent to solving the other in $L(G)$. Even when these are not exactly equivalent, the relationship can help us understand both problems better.

For example, as we see in the case of the eight rooks problem, finding a matching in a graph G is equivalent to finding an independent set in $L(G)$. A matching in G is a set of edges such that no two edges share an endpoint. A set of edges in G is precisely a set of vertices in $L(G)$, and sharing an endpoint is precisely what defines adjacency in $L(G)$. Therefore a set $M \subseteq E(G)$ is a matching in G if and only if it is an independent set in G.

The notation that we've used for the two problems is suggestive of this relationship: we used $\alpha(G)$ to denote the independence number of G, and $\alpha'(G)$ to denote the matching number. We could already have said the vague phrase, "$\alpha'(G)$ is the edge version of $\alpha(G)$". Now we can make this precise: $\alpha'(G) = \alpha(L(G))$.

Beyond this example, it is often the case that a numerical graph invariant $f(G)$ will be given an edge version $f'(G)$, defined as $f(L(G))$. Unfortunately, sometimes, an invariant will also be written $f'(G)$ if it gives off vague vibes of being the edge version of $f(G)$, even if it is not always equal to $f(L(G))$. You should be careful and check if $f'(G) = f(L(G))$ holds when faced with new notation; don't just assume it.

Question:	What do cliques in $L(G)$ correspond to?
Answer:	With one exception, a set of vertices in $L(G)$ is a clique when those vertices are edges in G that all share a common endpoint. If it weren't for that pesky exception, we could think of the maximum degree $\Delta(G)$ as being an "edge clique number" $\omega'(G)\ldots$

Question:	What's the pesky exception?
Answer:	There's another way for vertices in $L(G)$ to form a clique: if they correspond to the three edges of a cycle of length 3. Thus, if G is a graph with maximum degree 2, but one connected component of G is a cycle of length 3, then $\omega(L(G))$ will be 3, not 2.

The benefit of identifying the matching number $\alpha'(G)$ as $\alpha(L(G))$ is that it makes the problem of finding an independent set easier in a line graph. If we identify a graph H as being isomorphic to $L(G)$ for some G, then we can reduce the hard problem of finding $\alpha(H)$ to the easier problem of finding $\alpha'(G)$.

20.2 Euler tours and Hamilton cycles

An Euler tour in a graph is a closed walk that uses each edge exactly once. A Hamilton cycle in a graph is a cycle that uses each vertex exactly once. When the descriptions of two problems match so closely, but one uses "vertex" where the other uses "edge", it is natural to suspect that line graphs could be involved.

In fact, half of a relationship is present. We can prove the following result:

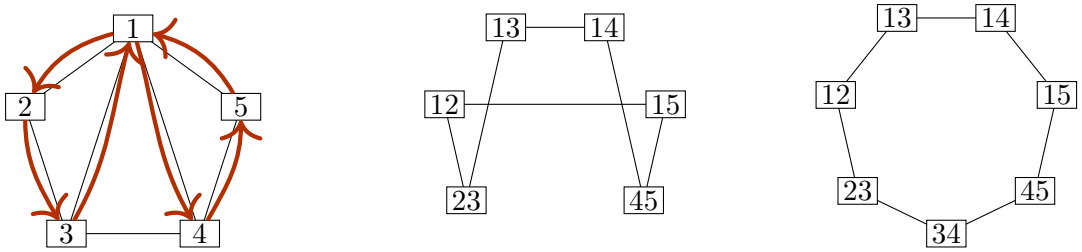
Proposition 20.1. *Every Euler tour in a graph G corresponds to a Hamilton cycle in $L(G)$.*

Proof. Suppose that the closed walk $(x_0, x_1, x_2, \ldots, x_m)$, with $x_m = x_0$, is an Euler tour in G. Then consider the following sequence of edges of G (or vertices of $L(G)$):

$$(x_0x_1, x_1x_2, x_2x_3, \ldots, x_{m-1}x_m, x_mx_0, x_0x_1).$$

These are, in fact, edges of G, because they are pairs of consecutive vertices of the closed walk. In fact, by the definition of an Euler tour, if we leave out the last element of this sequence (which is the same as the first element), then every edge of G is included exactly once. Finally, two consecutive edges in this sequence have the form xy, yz, and share an endpoint: so they are adjacent in $L(G)$. These are exactly the conditions needed to be certain that this sequence is a closed walk representing a Hamilton cycle in $L(G)$. □

Figure 20.3 shows a partial illustration of this principle, using as an example the graph G from Figure 20.2a, whose line graph was shown in Figure 20.2b. The closed walk in Figure 20.3a is



(a) A walk with no repeated edges (b) The corresponding 6-cycle (c) A Hamilton cycle in $L(G)$

Figure 20.3: Euler tours in G and Hamilton cycles in $L(G)$

not an Euler tour, but it is a closed walk with no repeated edges, so it shares most of the same properties. We can apply the same transformation

$$(1, 2, 3, 1, 4, 5, 1) \rightsquigarrow (12, 23, 13, 14, 45, 15, 12)$$

as we did in the proof of Proposition 20.1. (I have adjusted the order in which an edge is written, in some cases, so that it matches how Figure 20.3b is labeled.)

Question:	What can the argument of Proposition 20.1 tell us if we apply it to closed walks like this one, which use each edge at most once?
Answer:	The resulting closed walk in $L(G)$ represents a cycle, but not necessarily a Hamilton cycle. Here, the original walk did not use the edge 34, and so vertex 34 is not part of the cycle.

Figure 20.3c, on the other hand, shows a Hamilton cycle in $L(G)$. We should be suspicious of this, because the original graph G is not Eulerian: vertices 3 and 4 have odd degree! And, in fact, the converse to Proposition 20.1 is false: Hamilton cycles in $L(G)$ do not always correspond to Euler tours in G.

Question:	What went wrong?
Answer:	When visiting vertices $15, 14, 13, 12$ in $L(G)$, though they all share an endpoint, they all share the same endpoint. To continue a walk in G, it is not enough to know which edge was the last edge used; we need to know which of its endpoints was the last vertex used, to continue from that vertex.

This problem goes away if we consider directed graphs and their line digraphs. If a, a' are two arcs in a directed graph D, then the line digraph $L(D)$ can afford to be pickier about when it has an arc (a, a'). The arc (a, a') is not just included when a shares an endpoint with a', but when they share endpoints compatibly: when a ends where a' starts. This lets us prove the following:

Proposition 20.2. *If D is a directed graph, then Euler tours in D correspond to Hamilton cycles in $L(D)$, and vice versa.*

Proof. I will omit the proof that an Euler tour in D corresponds to a Hamilton cycle in $L(D)$, because the argument is the same as in Proposition 20.1.

Suppose that in $L(D)$, we have a Hamilton cycle represented by the walk

$$(a_0, a_1, \ldots, a_{m-1}, a_0).$$

For each $i = 0, 1, \ldots, m-1$, let $a_i = (x_i, y_i)$, and consider the sequence

$$T = (x_0, x_1, \ldots, x_{m-1}, x_0)$$

of vertices in D.

For each $i = 0, 1, \ldots, m-1$, there must be an arc (a_i, a_{i+1}) in $L(D)$, meaning that $y_i = x_{i+1}$. Therefore there is an arc (x_i, x_{i+1}) in D: this is another way to write the arc a_i. Similarly, there is an arc (a_{m-1}, a_0) in $L(D)$, so $y_{m-1} = x_0$, and therefore D has the arc (x_{m-1}, x_0). This means that T is a walk in D.

The arcs used by T are precisely the arcs $a_0, a_1, \ldots, a_{m-1}$, in that order. Since we started with a Hamilton cycle in $L(D)$, these arcs include each vertex of $L(D)$ exactly once, so they include each arc of D exactly once. Therefore T is an Euler tour. □

We solved the problem of finding Euler tours completely in Chapter 8, while the problem of finding Hamilton cycles is difficult and remains difficult in directed graphs. This means that if we spot that a certain directed graph happens to be a line digraph, we can make use of this to turn a difficult problem into an easy problem.

20.3 De Bruijn sequences

This happens, for example, in the study of de Bruijn sequences. To introduce these, let me give an example: the sequence

aabaacabbabcacbaccbbbcbccca,

which you should think of as being cyclic: when you get to the end, keep reading again from the start.[31] A 3-letter *substring* of this sequence is what you get if you start anywhere and read the next 3 letters: for example, you could start at the beginning, you will get aab, or start at the first c and get cab. There are 27 letters in the sequence, so there are 27 possible substrings; the magical property of this sequence is that every possible substring occurs exactly once.

Question:	Where does the substring aaa occur?
Answer:	You must start at the last character, reading an a and then reading aa from the beginning.

[31] Don't get stuck in an infinite loop, though. Eventually, stop reading the sequence and go back to reading about graph theory.

In general, given an alphabet A (any set of symbols) and a positive integer n, we can define the *de Bruijn sequence of order n* over A to be a cyclic sequence with the same property: its length-n substrings contain every possible substring exactly once. Of course, we can define anything we like, but that's no guarantee that such a thing exists!

These sequences are named after Nicolaas Govert de Bruijn, who proved in 1946 [11] that they exist: for all n and all alphabets A. To do this, de Bruijn used graph theory; but what is the right graph model for this problem?

One extreme option is to use A as the set of vertices, putting every possible edge between them, so that a de Bruijn sequence is a particular kind of closed walk in the graph. Such an approach is almost never the right choice, and it is not the right choice here. The problem is that the graph we get doesn't manage to encode the rules of the problem in any way: we don't benefit from looking at the problem with the graph in mind. Ideally, once we build the graph that describes the problem, the solution should be some very standard object inside that graph.

Another option is to take our vertices to be the set of all $|A|^n$ of the n-symbol strings that we want to see inside the de Brujin sequence. It is still tempting to have the de Bruijn sequence be a closed walk in the graph we define. To make this happen, we need the edges to answer the question: if string x is the n-symbol string at position i in the sequence, which strings could start at position $i+1$?

Question:	Suppose that reading from position i, we see the symbol $x_0x_1\dots x_{n-1}$. What strings could we see, reading from position $i+1$?
Answer:	We could see any of the sequences of the form $x_1x_2\dots x_n$, where $x_n \in A$ could be any symbol.

This is an asymmetric relationship, so we define a directed graph with all vertices of the form $x_0x_1\dots x_{n-1}$ and all arcs of the form $(x_0x_1\dots x_{n-1}, x_1x_2\dots x_n)$.

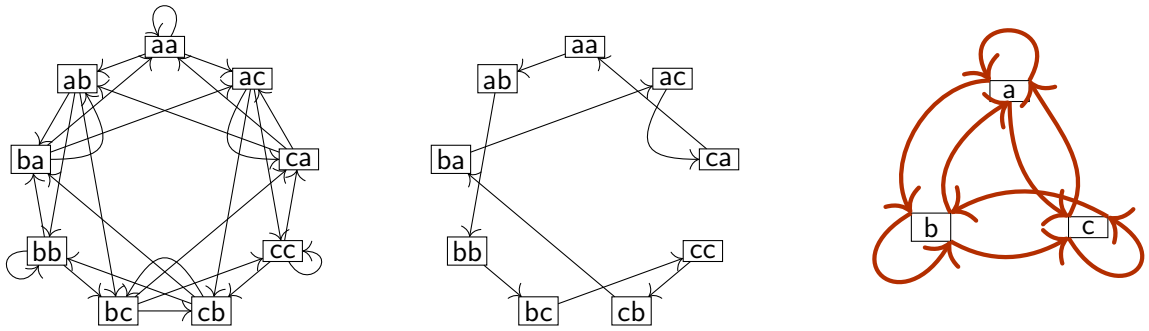
We call this directed graph the *de Brujin digraph*; the notation $B(k,n)$ is sometimes used for the de Bruijn digraph for strings of order n with a k-symbol alphabet. (Up to isomorphism of the graph, it does not matter what the alphabet is.) An example is shown in Figure 20.4a. I have kept $A = \{\mathsf{a},\mathsf{b},\mathsf{c}\}$, but to avoid making this graph too large, I have reduced n to 2.

Unfortunately, there's a problem. If we want to find a de Bruijn sequence of order n in the graph $B(k,n)$, we must find a Hamilton cycle in the graph, because we want to visit every vertex. Figure 20.4b shows such a cycle in $B(3,2)$. This gives us the de Bruijn sequence $\mathsf{aabbccbac}\dots$ but at what cost? Finding Hamilton cycles is very difficult, so we might as well have brute-forced the problem.

The trick is to realize the following incredibly convenient coincidence:

Proposition 20.3. *For all $k \geq 1$ and $n \geq 2$, the de Bruijn digraph $B(k,n)$ is isomorphic to the line digraph of $B(k,n-1)$.*

Proof. To find an isomorphism between $L(B(k,n-1))$ and $B(k,n)$, we need a function φ from the vertices of $L(B(k,n-1))$ (or the arcs of $B(k,n-1)$) to the vertices of $B(k,n)$.



(a) A diagram of $B(3,2)$ (b) A Hamilton cycle in $B(3,2)$ (c) An Euler tour in $B(3,1)$

Figure 20.4: Approaches to solving the de Bruijn sequence problems

An arc in $B(k, n-1)$ is a pair $(x_0x_1\ldots x_{n-2}, x_1x_2\ldots x_{n-1})$, where $x_0, x_1, \ldots, x_{n-1} \in A$. Since the vertices of $B(k,n)$ are also defined by n symbols from A, the natural thing to try is to define

$$\varphi\Big((x_0x_1\ldots x_{n-2}, x_1x_2\ldots x_{n-1})\Big) = x_0x_1\ldots x_{n-1}$$

and see if this works.

Question:	What does it mean for φ to work: what will make it an isomorphism?
Answer:	We need to check if φ preserves arcs: we want (x, y) to be an arc in $L(B(k, n-1))$ if and only if $(\varphi(x), \varphi(y))$ is an arc in $B(k,n)$.

Take two vertices in $L(B(k, n-1))$: let one vertex be the pair $(x_0x_1\ldots x_{n-2}, x_1x_2\ldots x_{n-1})$, and let the other be the pair $(y_0y_1\ldots y_{n-2}, y_1y_2\ldots y_{n-1})$. There is an arc from the first two the second exactly when $x_1x_2\ldots x_{n-1} = y_0y_1\ldots y_{n-2}$, by the definition of a line graph.

Now apply φ to both vertices: we get $x_0x_1\ldots x_{n-1}$ and $y_0y_1\ldots y_{n-1}$. By the definition of $B(n,k)$, there is an arc from the first to the second exactly when $x_1x_2\ldots x_{n-1} = y_0y_1\ldots y_{n-2}$: the same condition!

Therefore φ really is an isomorphism, completing the proof. □

Question:	How does Proposition 20.3 help us?
Answer:	Instead of a Hamilton cycle in $B(k,n)$, we can look for an Euler tour in $B(k, n-1)$, which is much easier.

For example, Figure 20.4c shows the same de Bruijn sequence aabbccbac, but this time it is obtained from an Euler tour in $B(3,1)$. (It's very hard to show Euler tours in a diagram; here, different visits to the same vertex touch it at different points. To obtain aabbccbac, start at the northeast corner of vertex a and follow the arrows.)

Theorem 20.4. *For any set of symbols A and any integer $n \geq 1$, there is a de Bruijn sequence of order n over A.*

Proof. For $n = 1$, just write all the symbols in A, one after the other.

For $n \geq 2$, we must prove that the digraph $B(k, n-1)$ is Eulerian, where $|A| = k$. By Corollary 8.6, we must check two properties of $B(k, n-1)$. (Both properties were defined in Chapter 8.)

First, we check that $B(k, n-1)$ is *weakly connected*. We can prove this by giving an $x - y$ walk for any two vertices $x = x_0x_1 \dots x_{n-2}$ and $y = y_0y_1 \dots y_{n-2}$. One possible walk passes through all vertices of the form

$$\underbrace{x_i x_{i+1} \dots x_{n-1} x_{n-2}}_{n-2-i \text{ symbols from } x} \underbrace{y_0 y_1 \dots y_{i-2} y_{i-1}}_{i \text{ symbols from } y}$$

for $i = 0, 1, \dots, n-2$, in that order.

Second, we check that all vertices of $B(k, n-1)$ are *balanced*, with indegree equal to outdegree. In fact, each vertex $x_0x_1 \dots x_{n-2}$ has both indegree and outdegree k: it receives an arc from every vertex of the form $xx_0x_1 \dots x_{n-3}$, where $x \in A$, and sends an arc to every vertex of the form $x_1x_2 \dots x_{n-2}x$, where $x \in A$.

Question:	Whenever you see that $n - 3$, you should be worried that $n - 3$ is negative. What happens then?
Answer:	Then, interpret $x_0x_1 \dots x_{n-3}$, and for that matter $x_1x_2 \dots x_{n-2}$, as being empty. When $n = 2$, each vertex receives an arc from every vertex of the form x, where $x \in A$; that is, from every vertex.

By applying Corollary 8.6, we obtain an Euler tour in $B(k, n-1)$, which corresponds to a Hamilton cycle in $B(k, n)$, which gives us a de Bruijn sequence of order n. □

20.4 Edge coloring

So far, we've used line graphs to draw connections between ideas we've already seen; now, let's use them to define something new: the edge chromatic number. This invariant is also sometimes called the "chromatic index"; I don't like this terminology, because there's nothing about the words "number" and "index" suggesting that one is about vertices and the other is about edges.

If G is a graph, we define the *edge chromatic number* $\chi'(G)$ to be $\chi(L(G))$: the chromatic number of the line graph of G. We can also avoid invoking $L(G)$, and instead define $\chi'(G)$ to be the least number of colors needed for an *edge coloring* of G. This is a function $f\colon E(G) \to C$, where C is a set of colors, such that each vertex is incident to at most one edge of each color.[32]

Question:	What are the color classes of an edge coloring?
Answer:	The color classes of a vertex coloring of G are independent sets in G, so the color classes of an edge coloring of G are independent sets in $L(G)$: these correspond to matchings in G.

[32] As in Chapter 19, this is really the definition of a *proper edge coloring*, but we will not consider any other kind.

The upper and lower bounds from the previous chapter can also be used here: we have $\omega(G) \leq \chi(G) \leq \Delta(G) + 1$ for all graphs G, so in particular, $\omega(L(G)) \leq \chi(L(G)) \leq \Delta(L(G)) + 1$.

Question:	What does $\omega(L(G)) \leq \chi(L(G))$ say, when translated from $L(G)$ back to G?
Answer:	We've already seen that $\omega(L(G))$ is $\Delta(G)$, with one exception, and $\omega(L(G)) \geq \Delta(G)$ even then. Therefore $\chi'(G) \geq \Delta(G)$.

Question:	What about the upper bound $\chi(L(G)) \leq \Delta(L(G)) + 1$?
Answer:	An edge xy shares an endpoint with at most $\Delta(G) - 1$ other edges at x, and $\Delta(G) - 1$ more at y; therefore xy has degree at most $2\Delta(G) - 2$ in $L(G)$. We can conclude that $\chi'(G) \leq 2\Delta(G) - 1$.

The inequalities $\Delta(G) \leq \chi'(G) \leq 2\Delta(G) - 1$ are intriguing: the lower and upper bound are both in terms of the maximum degree. However, much more will turn out to be true!

Although the definition of $\chi'(G)$ is new, we have already seen one very similar problem already. In Chapter 16, we looked at 1-factorizations, which are decompositions of a regular graph into perfect matchings.

Question:	If a k-regular graph G has a 1-factorization, what does that say about $\chi'(G)$?
Answer:	We can use the 1-factorization to get an edge coloring of G, by making each perfect matchings one of the color classes. There are k perfect matchings in the 1-factorization, so $\chi'(G) = k = \Delta(G)$: we already know this is the minimum possible.

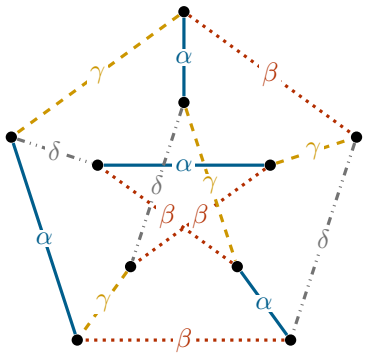
In Chapter 16, we proved two results about 1-factorizations. Theorem 16.3 says that for all even n, K_n has a 1-factorization; this means $\chi'(K_n) = n - 1$ when n is even. Also, Theorem 16.5 says that every k-regular bipartite graph G has a 1-factorization; this means $\chi'(G) = k$.

With just a little bit of trickery, we can make the second result more powerful. Let G be any bipartite graph with maximum degree $\Delta(G)$. There are many ways to find a bipartite graph H which contains G as a subgraph, and is $\Delta(G)$-regular. (I will leave it to you to discover how to do this, in a practice problem at the end of this chapter.) Then $\chi'(H) = \Delta(G)$, and so in particular $\chi'(G) = \Delta(G)$: inside every edge coloring of H, there is an edge coloring of G. We conclude:

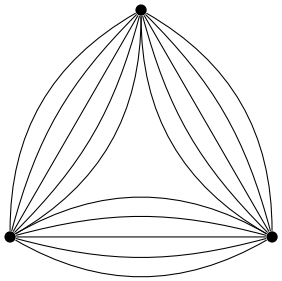
Corollary 20.5. *If G is any bipartite graph, then $\chi'(G) = \Delta(G)$.*

(This result, in somewhat different terminology, was proven by Kőnig in 1916 together with Theorem 16.5 [64].)

Thus far, we've seen many instances where $\chi'(G) = \Delta(G)$. Is this universal? No: for example, this will not happen for K_n when n is odd. In this case, the largest matching has only $\frac{n-1}{2}$



(a) An edge 4-coloring of the Petersen graph



(b) All edges must have different colors

Figure 20.5: Two examples of lower bounds on edge coloring

edges, so it takes at least n matchings to cover all $\frac{n(n-1)}{2}$ edges of K_n. However, Corollary 16.4 says that when n is odd, K_n has a decomposition into n nearly perfect matchings; this means $\chi'(K_n) = n$ when n is odd.

Another interesting example is the Petersen graph. It is 3-regular, and has many perfect matchings, so you might be tempted to hope that it has edge chromatic number 3. However, this is not true. Suppose for contradiction that the Petersen graph has an edge coloring with 3 colors $\{\alpha, \beta, \gamma\}$. (It is traditional to use Greek letters for the colors of edges; this has nothing to do with the graph invariants $\alpha(G)$ or $\beta(G)$.)

To color all 15 edges, each color class must contain 5 edges and be a perfect matching. If we take the subgraph formed by the edges with colors α and β, it is 2-regular: each connected component is a cycle. Moreover, the edges around a cycle must alternate between the colors α and β, so each component has even length.

There are two ways to have a 10-vertex graph whose connected components are cycles of even length: it can be a copy of C_{10}, or the disjoint union of copies of C_4 and C_6. In the Petersen graph, neither is possible: the first case contradicts Proposition 17.1, where we proved that the Petersen graph does not have a Hamilton cycle, and the second case contradicts Lemma 5.3, where we proved that it has no cycles of length 3 or 4. So the Petersen graph cannot have an edge coloring with only 3 colors.

The Petersen graph does have an edge coloring with 4 colors, as shown in Figure 20.5a. It is another example of a graph G with $\chi'(G) = \Delta(G) + 1$.

Is this the limit? Not if we consider multigraphs. Figure 20.5b shows an example given by Claude Shannon [91]; in this multigraph, any two edges share an endpoint, so they must all be given different colors. If there are k copies of each edge of the triangle, then the graph has maximum degree $2k$, but edge chromatic number $3k$ (equal to the number of edges). Shannon also proved that this is the worst case: $\chi'(G) \leq \frac{3}{2}\Delta(G)$, even if G is a multigraph.

For simple graphs, however, we have only seen examples where $\chi'(G)$ is either $\Delta(G)$ or $\Delta(G)+1$. In 1964, Vadim Vizing proved that this is true for all simple graphs G [102]. Since there are only two possible values of $\chi'(G)$, sometimes a graph G will be referred to as "class one" if $\chi'(G) = \Delta(G)$ and "class two" if $\chi'(G) = \Delta(G) + 1$.

I will conclude this chapter with a proof of Vizing's theorem:

Theorem 20.6 (Vizing's theorem)**.** *For all graphs G, $\chi'(G) \leq \Delta(G) + 1$.*

20.5 Vizing's theorem

All proofs of Vizing's theorem that I am aware of rely on a variant of the alternating paths we used in Chapter 14 or Chapter 16 when working with matchings. This makes sense, since a color class in an edge coloring of G is an independent set in $L(G)$: a matching in G. However, instead of trying to improve a single matching, we will be trying to rearrange the existing matchings to make room for one more edge.

Suppose that α and β are two of the colors used in an edge coloring of a graph. The subgraph formed by edges of colors α and β is the union of two matchings, so its connected components consist of paths and cycles. A path component of this subgraph is called an *α/β-path*.

The colors of the edges of an α/β-path alternate between α and β. If x is the start or end of an α/β-path, then only one of the two colors is used on edges incident to x, and it is the edge beginning the α/β-path. Conversely, if only one of α or β is used on edges incident to some vertex x, then we can find an α/β-path starting at x, by greedily following edges of color α or β until we cannot continue.

The use of an α/β-path is that we can swap the colors α and β on every edge of the path and obtain a different edge coloring. We can hope to use such a swap to make color α or β available for an edge we have not yet colored.

The proof that follows is based on a proof of Ehrenfeucht, Faber, and Kierstead [26], simplified from their more general statement.

Proof of Theorem 20.6. For a fixed number of colors q, we induct on the number of vertices in G, proving that if $\Delta(G) \leq q - 1$, then G has an edge q-coloring. We can take a 1-vertex graph as our base case, in which case no edges need to be colored at all. To induct, we take a graph G with $\Delta(G) \leq q - 1$ and an arbitrary vertex x, and assume that $G - x$ has an edge q-coloring; we must find a way to turn it into an edge q-coloring of G.

As we color the edges incident to x one at a time, we keep a partition of these edges into three sets $S \cup T \cup U$, each with their own meaning:

- Edges in S are *settled*, and their color will not change.
- There is at most one edge in T, which is *tentatively* colored.
- Edges in U are *uncolored*.

Our algorithm will maintain the following invariant at all times: the colors on $E(G - x) \cup S \cup T$ must always be an edge coloring of $G - U$.

When we begin, we put all edges incident to x in U. These edges have a valuable flexibility: in $L(G)$, they have at most $\Delta(G) - 1$, or $q - 2$, neighbors that have been given a color, so they have at least 2 colors available to them. To preserve this flexibility, we select two *candidate colors* for each edge $xy \in U$, satisfying a second invariant. For as long as edge xy remains in U, it will have two candidate colors, which must not appear on edges incident to y, nor on edges in S; we allow them to appear on an edge in T, since it is colored only tentatively.

Given this setup, there are several cases where we can make quick progress:

- First, if $T = \varnothing$, choose an arbitrary edge in U; give it one of its candidate colors, and move it to T. Otherwise, we will assume $T = \{xy\}$ and let α be the color of edge xy.
- Second, if α is not a candidate color of any edge in U, move xy from T to S. If α is a candidate color of only one edge $xz \in U$, we still move xy from T to S, but also move xz from U to T, giving xz its other candidate color.
- Third, if some edge $xz \in U$ has a color $\beta \neq \alpha$, and no other edge in U has β as one of its candidate colors, give edge xz color β and move it from U to S directly.

If none of these apply, then every color that appears as a candidate color in U (which includes α) must appear at least twice. Every edge in U only has two candidate colors, so at most $|U|$ colors appear as candidate colors at all. At most $|S| + |U|$ colors appear on edges incident to x in any fashion: as a color on any edge, or as a candidate color. But $|S| + |U| \leq \Delta(G) - 1$, and $q = \Delta(G) + 1$, so we can pick a color γ different from all of these: not used on any edge in $S \cup T$, nor as a candidate color of any edge in U.

Let P be the α/γ-path starting at x (with xy as its first edge), and swap the colors α and γ along P.

Question:	Can this swap interfere with the colors of edges in S, or with the candidate colors of edges in U?
Answer:	Edges in S cannot have color α or γ, so they are untouched by the swap. An edge $xz \in U$ might have α as one of its candidate colors. If so, P might visit vertex z, but it will have to stop there, because z is not incident to any edges of color α.

Suppose this last possibility occurs: the path P ends at a vertex z such that $xz \in U$, and α is a candidate color of z. (The last edge of P must have had color γ, swapped to α.) Then we make one further modification: we replace α by γ as a candidate color of xz. After all, α is no longer available for z, but γ is.

Whether or not this happens, γ (the new color of xy) appears at most once as a candidate color, so we return to one of the cases where we can make quick progress. This lets us keep going until all edges are in S and the edge coloring of G is complete.

Question:	How do we know that we eventually reach this point?
Answer:	In each of the cases where we make quick progress, we move an edge "earlier in the alphabet": from T to S, or from U to T, or from U directly to S. This can only stop when all edges are in S.

This completes the induction step, showing that if $\chi'(G - x) \leq q$, then $\chi'(G) \leq q$ as well. By induction, $\chi'(G) \leq q$ for all graphs G with $\Delta(G) \leq q - 1$, proving Vizing's theorem. $\square$

20.6 Practice problems

1. A graph G is called claw-free if it does not have a copy of the star graph S_4 as an induced subgraph. In other words, no vertex $x \in V(G)$ can have three neighbors y_1, y_2, y_3 with no edges between them.

   Prove that all line graphs are claw-free graphs.

2. Use an Euler tour to find a de Bruijn sequence of order 4 over the alphabet $A = \{0, 1\}$.

3. Prove that there is a length 99 cyclic sequence of 0's, 1's, and 2's such that among the substrings 00, 01, 02, 10, 11, 12, 20, 21, and 22, there is one that occurs 0 times in the string, one that occurs 1 time, one that occurs 2 times, one that occurs 10 times, one that occurs 11 times, one that occurs 12 times, one that occurs 20 times, one that occurs 21 times, and one that occurs 22 times.

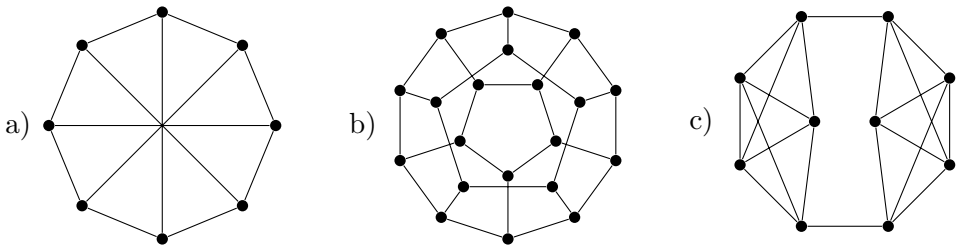
4. The line graph $L(K_{3,3})$ shown in Figure 20.2d has an unusual property: it is isomorphic to its own complement.

   a) Find one such isomorphism.

   b) Find an 8-vertex graph with the same property.

   c) Prove that there is no 10-vertex graph with this property.

5. Find the edge chromatic number of the following graphs:



6. To complete the proof of Corollary 20.5, we need to show that every bipartite graph G is a subgraph of a $\Delta(G)$-regular graph H.

   a) Suppose that G has a bipartition (A, B) with $|A| = |B|$. Prove that it is possible to add only edges to G, and no new vertices, to obtain a $\Delta(G)$-regular multigraph H.

   (How can you make use of this result if $|A| \neq |B|$?)

   b) Prove that if $\delta(G) < \Delta(G)$, then it is possible to add edges between two copies of G to obtain a bipartite graph G' with $\Delta(G') = \Delta(G)$ but $\delta(G') = \delta(G) + 1$.

   Conclude that there is a $\Delta(G)$-regular bipartite graph H containing G which has $2^{\Delta(G)-\delta(G)} \cdot |V(G)|$ vertices.

7. More recent proofs of Vizing's theorem, like the one in this chapter, are usually written so that they can also prove stronger claims. Here are two such claims.

   a) Modify the proof to show that if the vertices in G which have degree $\Delta(G)$ form an independent set, then $\chi'(G) = \Delta(G)$.

b) Modify the proof to show that if the subgraph of G induced by the vertices of degree $\Delta(G)$ is a forest, then $\chi'(G) = \Delta(G)$.

8. (BMO 1992) The edges of a connected n-vertex graph G are colored red, blue, and yellow so that each vertex has one incident edge of each color.[33]

   a) Prove that n must be even and that for all even $n > 2$, a graph with such an edge coloring exists.

   b) Suppose that for some subset $S \subseteq V(G)$, there are r red, b blue, and y yellow edges with exactly one endpoint in S. Prove that r, b, y are either all even or all odd.

[33] I have paraphrased; the original problem on the British Mathematical Olympiad had an elaborate and very long statement about dwarfs and orcs.

Planar Graphs

21 Planar graphs

The purpose of this chapter

This chapter is an introduction to the ideas you'll need to learn about planar graphs. Most applications of graph theory to geometry are based on Euler's formula and the face length formula, which you will see in this chapter.

The toughest decision for me in presenting this material was to decide how much of the technical geometry to include. The truth is that when working with planar graphs, we don't want to, and also don't have to, worry about these details. The standard academic solution is to begin with a section of technical preliminaries. However, this is also a good way to bore all my readers before we get to the good part.

My compromise was to include rigorous technical proofs of the geometric claims in the last section of this chapter, for the interested reader only. Usually, I cite earlier theorems and lemmas in the textbook when I use them, but in this case, I won't, so that if you don't want to think about these details, you won't have to. However, I have taken care to make sure that every argument in this part of the textbook can be made rigorous using one of the claims in this section.

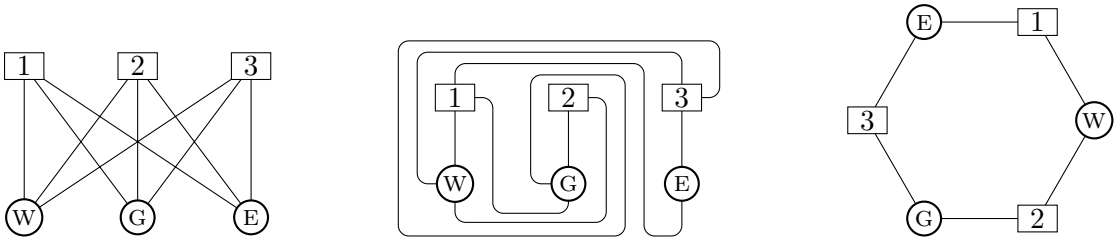
21.1 Three utilities

Some puzzles are invented because they have a clever solution. Other puzzles are invented because they have no solution; I suspect that sometimes, the motivation is to keep a clever child quietly working on the puzzle for a long time without worries that the child will actually ever solve the puzzle and bother you again.

A classic entry in this genre of puzzles is the three utilities problem [66]. Here, three houses (which we will number 1, 2, and 3) need to be provided water, gas, and electricity by lines from three central utilities (which we will label W, G, and E). The required connections are shown in Figure 21.1a; however, this diagram is not a solution because (for some reason that's never been clear to me) water lines, gas lines, and electricity lines cannot cross. Perhaps the utility providers in this town have not yet learned about the third dimension.

In any case, there is no way to solve the problem. Many attempts look like the diagram in Figure 21.1b; here, curved paths that circle around each other in complicated ways may confuse even the solver into forgetting that house 2 still does not have electricity. (The inhabitants of house 2 are presumably very clear on this point, however.)

But why do all attempts fail? Before presenting you with any general theory, let me give a specific argument for the utility graph. (This is not the most common argument, and [66]



(a) The utility graph (b) Not a solution (c) A mandatory hexagon

Figure 21.1: The three utilities problem

contains another simple solution. I follow the same approach as West's *Introduction to Graph Theory* [104], and for the same reason: it is a special case of the overlap graph technique we will see in Chapter 22.)

Question:	The utility graph shown in Figure 21.1a is isomorphic to another graph we have a name for; what is it?
Answer:	We have a couple of names, actually! The utility graph is isomorphic to the complete bipartite graph $K_{3,3}$; it is also isomorphic to the circulant graph $\mathrm{Ci}_6(1,3)$.

The utility graph is Hamiltonian; one possible Hamilton cycle is represented by

$$(1, \textsc{w}, 2, \textsc{g}, 3, \textsc{e}, 1).$$

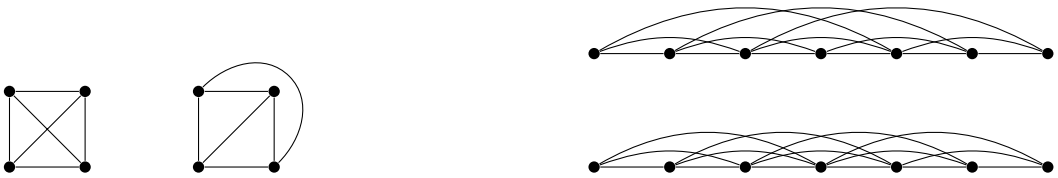
No matter how this cycle is drawn in a hypothetical solution, it will be a closed loop in the plane with an inside and an outside. (This is even true in Figure 21.1b, though the closed loop is rather complicated!) To help us visualize the closed loop, I will draw it as a regular hexagon in Figure 21.1c, though of course the loop doesn't have to look anything like this.

There are still three edges that we have not drawn: the edges 1G, 2E, and 3W. Only one of these edges can be drawn inside the loop. For example, if we draw edge 1G inside the loop, then it separates vertices 2 and W from vertices 3 and E, so neither 2E nor 3W can also be drawn inside the loop.

Of course, we can still draw either of those edges outside the loop. However, an identical problem occurs there. Suppose we decide to draw edge 2E outside the loop. No matter how we do it, the edge and the loop divide the outside into two pieces: one (call this F_1) bounded by the edges $\{2\textsc{e}, 2\textsc{g}, 3\textsc{e}, 3\textsc{g}\}$ and one (call this F_2) bounded by the edges $\{1\textsc{e}, 1\textsc{w}, 2\textsc{e}, 2\textsc{w}\}$. One of these pieces will be finite and the other infinite, but that doesn't concern us in any way.

What does concern us is that once we've drawn edge 2E outside the loop, we can't do the same thing with edge 1G or 3W. A curve in the plane from 1 to G drawn outside the loop, for example, would start in F_1 and end in F_2, so at some point it would have to cross one of the boundaries.

Of course, neither drawing 1G inside the loop nor drawing 2E outside the loop is required, but we're stuck no matter what we try: the loop has two sides (inside and outside) and each side only has room for one edge, but we have three edges left to draw. Because $3 > 2$, the three utilities problem has no solution.



(a) Two drawings of K_4

(b) Which graph is planar?

Figure 21.2: Planar and non-planar graphs

21.2 Planar graphs

The three utilities problem is one special case of a general problem in graph theory: which graphs can be drawn in the plane without crossings?

Sometimes answering this question is easy. For example, Figure 21.2a shows two ways to draw the complete graph K_4. In the standard one, the two diagonal edges cross; however, we can draw one of the two edges differently and eliminate that crossing.

Sometimes answering this question is hard. Both of the graphs in Figure 21.2b are drawn with many crossing edges. For one of them, this can be fixed; for the other, there is no way to avoid crossings. However, it is far from obvious which graph has which property; we will need to learn more before we answer this question.

Question:	In Figure 21.2a, we avoid crossings by curving one of the edges. However, we can also draw K_4 in the plane with 6 straight edges that don't cross; how?
Answer:	Put 3 vertices of K_4 at the corners of a triangle, and put the 4^{th} vertex inside the triangle.

It's important to distinguish between two similar concepts: "a graph that can be drawn in the plane with no crossings" and "a drawing of a graph in the plane with no crossings." The first of these is a graph invariant. If G and H are two isomorphic graphs, and we can draw G in the plane with no crossings, then we can draw H in the plane with no crossings: just relabel the drawing of G.

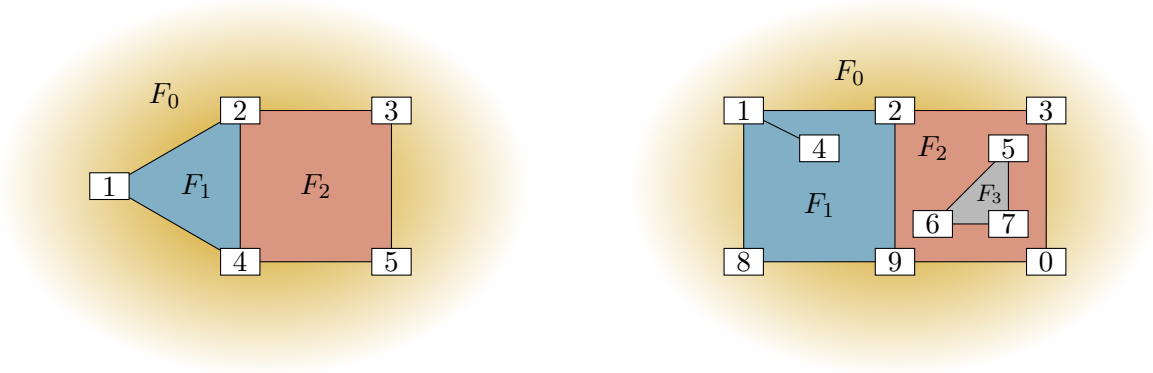
However, whether or not a graph is drawn with no crossings is not a graph invariant! Figure 21.2a makes this clear: the graph K_4 can be drawn with crossings, but it can also be drawn without crossings, without changing the graph itself.

Accordingly, we have two definitions to separate these concepts.

Definition 21.1. *A **plane embedding** of a graph G is a drawing of G in the plane with no crossings.*

*A **planar graph** is a graph that has a plane embedding.*

This means that our discussion of the the three utilities problem can be taken as a proof of the following proposition:



(a) Simple faces

(b) More unusual faces

Figure 21.3: Examples of faces in plane embeddings

Proposition 21.1. *The complete bipartite graph $K_{3,3}$ is not planar.*

Let me emphasize again that a planar graph and a plane embedding are two different things. A planar graph is still just an abstract object: a set of vertices and a set of edges. We haven't picked a particular drawing for it, and there could be many drawings that are different from each other in important ways.

When using the assumption that a graph G is planar in a proof, we should usually begin by choosing a plane embedding of G. However, we should be careful when talking about properties of that plane embedding: they are not properties of the graph G itself, unless we can prove that all plane embeddings of G share those properties. In this chapter, we will see some examples where we can prove this, and some examples where we can't.

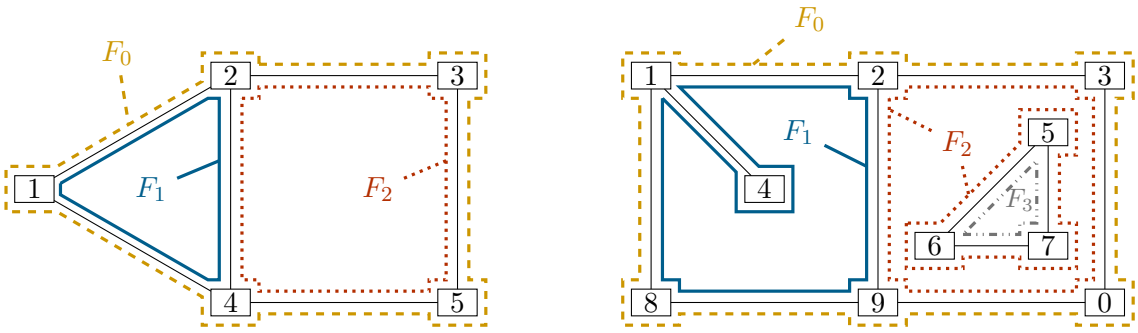
We will occasionally have reason to talk about plane embeddings of multigraphs, rather than graphs. This will not come up when deciding if a graph is planar or not: a multigraph is planar if and only if its simplification is planar. But we sometimes want to consider plane embeddings of multigraphs as objects of study in their own right.

21.3 Faces

The diagrams in Figure 21.3 show two plane embeddings in which I've highlighted several regions of the plane. (The shading in regions that I've labeled F_0 in both diagrams is meant to indicate that they are infinite regions, extending outward to the rest of the plane.) We call these the faces of the plane embedding.

Definition 21.2. *The **faces** of a plane embedding are the connected regions of the plane separated by the edges of the plane embedding. One of the faces is infinite, and contains all points sufficiently far from the plane embedding; it is called the **outer face**.*

Faces are the link between the geometry of a plane embedding and the abstract structure of the graph G. In the graph G itself, a region of points does not exist. So what does? What's left is the boundary of the face: the vertices and edges that separate it from the other faces.



(a) Simple faces

(b) More unusual faces

Figure 21.4: Boundary walks of the faces in Figure 21.3

In ideal circumstances, the boundary of a face is a cycle, in which case (as usual) it is represented by a closed walk. Don't forget that the cycle doesn't have a prescribed starting point or direction, so it can be represented by many different closed walks.

Question:	In Figure 21.3a, which closed walks represent the boundary of the faces?
Answer:	The boundary of F_1 is represented by $(1, 2, 4, 1)$, the boundary of F_2 is represented by $(2, 3, 5, 4, 2)$, and the boundary of F_0 is represented by $(1, 4, 5, 3, 2, 1)$.

However, the boundary of a face is not always as nice. In Figure 21.3b, the boundaries of faces F_1 and F_2 are a bit more unusual:

- In the case of F_1, we would like edge 14 to be taken into account, even though it only separates face F_1 from itself, so the boundary is not a cycle. However, the closed walk $(1, 2, 9, 8, 1, 4, 1)$ still seems to represent the boundary.
- In the case of F_2, the boundary is not even connected! We need two closed walks: $(2, 3, 0, 9, 2)$ and $(5, 6, 7, 5)$.

In general, the boundary of a face is represented by one or more **boundary walks**. A boundary walk is found by starting at any vertex on the boundary of the face, and following edges out of it in some direction, while staying inside the face. Figure 21.4 shows the boundary walks of the faces of the two example graphs we looked at.

Question:	Looking at Figure 21.3b, why does face F_2 have two boundary walks?
Answer:	The graph has multiple connected components, each of which contributes a boundary walk to F_2.

Question:	Why is edge 14 in Figure 21.3b on the boundary walk of F_1 twice?
Answer:	The same face, F_1, is on both sides of the edge.

Every edge xy is either on two boundary walks of two different faces, or on the boundary walk of the same face twice. The difference between these two cases comes down to the following test:

Lemma 21.2. *If edge xy is a bridge of planar graph G, then in every plane embedding of G, it appears on the boundary walk of the same face twice.*

If edge xy is not a bridge, then in every plane embedding of G, it separates two faces, and appears once on the boundary walk of each face.

Proof. If the same face F is on both sides of edge xy when it is drawn in the plane embedding, then we can draw a curve from one side of xy to the other while always staying inside F. If xy is deleted, we can complete that curve to a closed loop drawn entirely inside F. Of the two vertices x and y, one is drawn inside that closed loop and the other outside it, and no edges cross the loop; therefore there is no way to get from x to y. This makes xy a bridge.

If edge xy separates two faces F_1 and F_2, then each of them has a boundary walk using edge xy once. Without loss of generality, such a boundary walk goes from x to y, and returns along some $y - x$ walk without using edge xy again: that walk exists in $G - xy$. Every $y - x$ walk contains a $y - x$ path (by Theorem 3.1), so there is an $x - y$ path in $G - xy$. This means that in G, there is a cycle containing xy, so xy is not a bridge.

The above arguments tell us whether xy is a bridge or not based on what it does in one plane embedding. But regardless of the plane embedding, edge xy either is a bridge in graph G or it isn't, so it must have the same behavior in every plane embedding. □

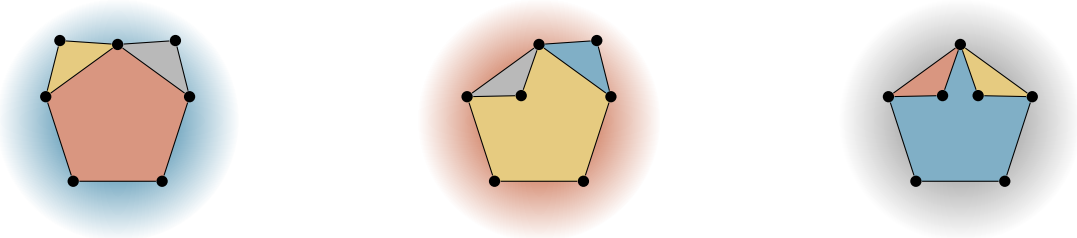
The most important property of the boundary of a face is its length. The **length** of a face F, written $\text{len}(F)$, is the sum of the lengths of boundary walks of F. Equivalently, $\text{len}(F)$ is the number of edges on the boundary of F, counting an edge twice if it is not on the boundary of any other face.

When we give the definition of $\text{len}(F)$ in its second form, it seems unnatural and artificial, though you may remember that when we defined the degree of a vertex of a multigraph in Chapter 7, we did something similar. Just like back then, it is all in service of making sure a nice lemma holds in all possible cases:

Lemma 21.3 (Face length formula)**.** *If G is a planar graph with m edges, and a plane embedding of G has faces $F_0, F_1, \ldots, F_{r-1}$, then*

$$\sum_{i=0}^{r-1} \text{len}(F_i) = 2m.$$

Proof. Each edge of G appears once on boundary walks of two faces, or twice on the boundary walks of one face. In either case, it contributes $+2$ to the sum of face lengths, so the contributions of all m edges add up to $2m$. □



(a) (b) (c)

Figure 21.5: Three embeddings of the same graph

The face length formula tells us that the sum of the lengths of the faces does not depend on the plane embedding chosen. (After all, it is equal to $2m$, which certainly cannot change based on the plane embedding!) The individual lengths, however, certainly do depend on the embedding!

Figure 21.5 shows three different plane embeddings of the same 7-vertex graph. (I have deliberately not used consistent colors to discourage you from the temptation of feeling that the faces of one embedding correspond in any way to the faces of another.) However, the face lengths vary:

- The embedding in Figure 21.5a has two faces of length 3, one face of length 5, and an outer face of length 7.
- The embedding in Figure 21.5b has two faces of length 3, and two faces (including the outer face) of length 6.
- The embedding in Figure 21.5c is almost like the one in Figure 21.5a: it also has faces of lengths $3, 3, 5, 7$. However, in this case, the embedding of length 5 is the outer face.

21.4 Euler's formula

The face length formula is one of two important identities regarding the faces of a plane embedding. The other one is Euler's formula (one of several mathematical statements by that name!), which helps us count the faces in a plane embedding. You may have noticed that no matter how hard we tried to find different plane embeddings in Figure 21.5,all of them had four faces. This is not a coincidence, and it will follow from Euler's formula that the number of faces in a plane embedding depends only on the graph.

Theorem 21.4 bears Euler's name because he was the first to look at the problem in a general case [32]. Euler did not consider the problem for planar graphs, but for polyhedra, which we will study in Chapter 23. Many other mathematicians later approached the formula from different directions; see [27] for a brief history, as well as further references and many different proofs.

Euler's formula is often stated as $V - E + F = 2$, where V is the number of vertices, E is the number of edges, and F is the number of faces. I can't bring myself to do this, since I think of V and E as sets, not numbers. In graph theory, n and m are the standard variables to use when you want to count the vertices and edges in a graph, respectively. There is no standard

variable for faces, so I will use r (for "region"). Meanwhile, k will be our variable of choice for connected components, as it already was in Chapter 10.

Theorem 21.4 (Euler's formula)**.** *If a connected plane embedding (of a graph or a multigraph) has n vertices, m edges, and r faces, then*

$$n - m + r = 2.$$

In general, if there are k connected components, this formula becomes

$$n - m + r - k = 1.$$

Proof. We induct on the number of edges, m. Since removing an edge from a graph may disconnect it, we should work directly with the second, more general form of Euler's formula; the first form will follow by setting $k = 1$.

Our base case is $m = 0$. Here, n (the number of vertices) is equal to k (the number of connected components), and there is only one face: $r = 1$. So $n - m + r - k$ starts out at 1.

Now assume for some $m \geq 1$ that Euler's formula holds for all plane embeddings with $m - 1$ edges, and consider a plane embedding of a graph G with m edges. We will arbitrarily pick an edge xy to delete.

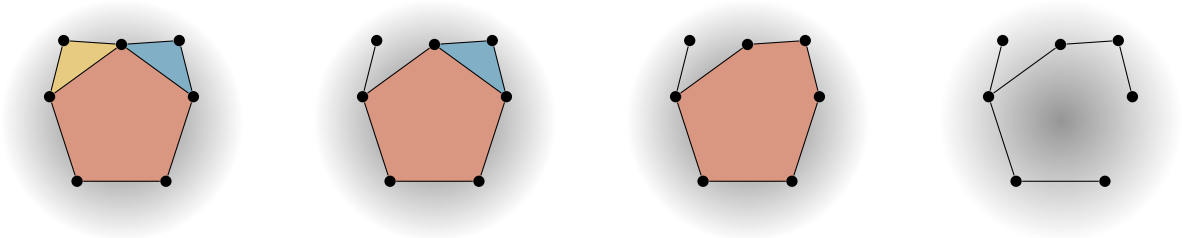
Question:	How can the deletion of edge xy affect n, m, r, and k?
Answer:	It never affects n, and always decreases m by 1.
	It may decrease r by 1, if xy previously separated two faces.
	It may increase k by 1, if xy was the only way to get from x to y.

We would like $n - m + r - k$ to stay the same, so we would like exactly one one of two things to happen: either r decreases by 1, or k increases by 1, but not both.

Fortunately, this is exactly what Lemma 21.2 tells us! If edge xy is a bridge, its deletion increases k by 1; however, in every plane embedding, the same face is on both sides of xy, so r stays the same. If edge xy is not a bridge, then it separates two faces, so deleting it decreases r by 1; however, it is not a bridge so its deletion does not affect r.

In summary: the number of vertices, edges, faces, and components in the plane embedding of $G - xy$, which we'll denote (n', m', r', k'), is either $(n, m-1, r-1, k)$ or $(n, m-1, r, k+1)$. By the induction hypothesis, $n' - m' + r' - k' = 1$, which tells us either that $n - (m-1) + (r-1) - k = 1$ or that $n - (m-1) + r - (k+1) = 1$; both of these simplify to $n - m + r - k = 1$.

By induction, the formula holds for all plane embeddings. You can see the induction in action in Figure 21.6. Here, we've chosen to delete only edges that keep the graph connected, so going from Figure 21.6a to Figure 21.6d, r decreases at each step while k remains equal to 1. At this point, if we delete more edges, it will decrease m and increase k while keeping n and r the same. □



(a) $(n, m, r) = (7, 9, 4)$ (b) $(n, m, r) = (7, 8, 3)$ (c) $(n, m, r) = (7, 7, 2)$ (d) $(n, m, r) = (7, 6, 1)$

Figure 21.6: How the triple (n, m, r) changes as we erase edges in a plane embedding

In Euler's formula, the variables n, m, and k are properties of the graph G, while r is computed from a specific plane embedding. However, we can solve for r in terms of n, m, and k: it is $m - n + k + 1$, or just $m - n + 2$ if G is connected.

Question:	How can these observations be reconciled?
Answer:	It means that although different plane embeddings can have faces with different shapes and with different lengths, two plane embeddings of the same graph are guaranteed to have the same number of faces: the number predicted by Euler's formula.

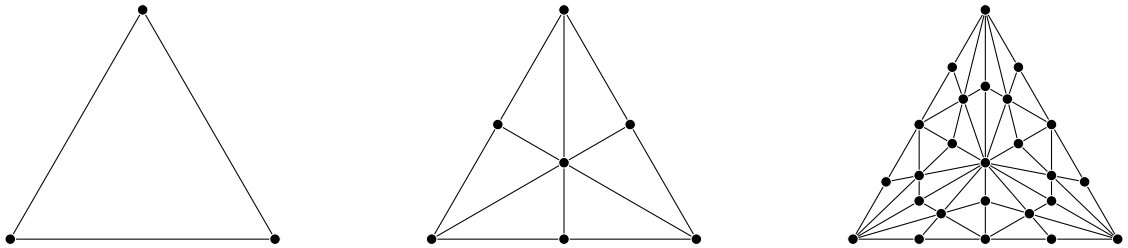
We should still avoid talking about "the faces of G", because that is not a well-defined concept, as Figure 21.5 shows. Talking about the number of faces that G has is also incorrect, but if you said it to my face, I would only disapprove a little; after all, if we pick different embeddings, we will still not disagree about the number of faces.

21.5 Barycentric subdivisions

Euler's formula and the face length formula have applications to geometry in scenarios where the problem can be modeled as a graph. To illustrate this, let me tell you about a problem that once misled me until I thought of applying Euler's formula to solve it.

The *barycentric subdivision* of a triangle subdivides it into six triangles, by drawing the three medians: the lines connecting each corner of the triangle to the midpoint of the opposite sides. (The three lines in this way always intersect at a single point, which is called the centroid of the triangle.) An example is shown in Figure 21.7b, though the triangle we start with does not have to be equilateral.

We can iterate this process: start with the barycentric subdivision of a triangle, and then take the barycentric subdivision of each of the six small triangles that result. This is shown in Figure 21.7c, but we don't have to stop there; we can keep going, by taking the barycentric subdivision of each of the 36 small triangles in that diagram. Let's say that stage n of this process is the result we get when we apply the operation "take the barycentric subdivision of every triangle which does not have any smaller triangles inside it" n times.



(a) Stage 0 (b) Stage 1 (c) Stage 2

Figure 21.7: Iterated barycentric subdivisions

I was interested in the number of vertices in this diagram: I placed a vertex at every point where two or more segments met, or in other words, at every point which is the corner of one or more triangles. If you count the vertices in Figure 21.7, you get 3 vertices in Figure 21.7a, 7 vertices in Figure 21.7b, and 25 vertices in Figure 21.7c. I went one step further and laboriously computed the number of vertices in Stage 3, which turned out to be 121.

I did not even want to think about drawing Stage 4, so my next step was to go to the Online Encyclopedia of Integer Sequences and search for the sequence $3, 7, 25, 121$. At the time, there was essentially one candidate formula in the search results (plus a few minor variants which were clearly wrong for this problem), but it was a very nice formula: it suggested that the n^{th} stage would contain $(n+2)! + 1$ vertices.

Question: At this point, there is a quick way to see that this formula can't possibly be right for all n, because the number of vertices grows at most exponentially. Why?

Answer: The number of triangles in the subdivision is much easier to count: it is 6^n in the n^{th} stage, because each triangle gets subdivided into six triangles. Each triangle has 3 corners, giving $3 \cdot 6^n$ vertices total. This counts some vertices many times, because some vertices are the corners of many triangles; however, it counts each vertex at least once, so there are at most $3 \cdot 6^n$ vertices.

Unfortunately, I only thought of this quick way later. At the time, I spent a while trying to prove the formula $(n+1)! + 1$ combinatorially. Maybe we should label the vertices with permutations of $n+2$ elements, but leave the central vertex unlabeled? In the end, it occurred to me to try Euler's formula, and that's when my house of cards fell apart.

Why Euler's formula? Well, the diagrams shown in Figure 21.7 and the ones that come after are all plane embeddings: the vertices are exactly as I have defined them already, and the edges are the line segments connecting the vertices. We can hope that the edges and faces will be easier to count than the vertices, replacing one hard counting problem with two easier ones.

In this case, to apply Euler's formula, we should start with the faces. It is not quite true that there are 6^n faces, all with 3 sides, because there is also an outer face; there are $6^n + 1$ faces total.

Question:	What is the length of the outer face?
Answer:	It is $3 \cdot 2^n$: in the n^{th} stage, each side of the triangle has been divided into 2^n segments, which are all edges of the plane embedding.

Question:	How can we find the number of edges?
Answer:	We can add up the lengths of the faces and apply the face length formula.

With 6^n faces of length 3 and one face of length $3 \cdot 2^n$, the sum of face lengths is $3 \cdot 6^n + 3 \cdot 2^n$, so there $\frac{3 \cdot 6^n + 3 \cdot 2^n}{2}$ edges. Solving for the number of vertices in Euler's formula, we get

$$\frac{3 \cdot 6^n + 3 \cdot 2^n}{2} - (6^n + 1) + 2,$$

which simplifies to $\frac{6^n + 3 \cdot 2^n}{2} + 1$. When $n = 4$, the first case I did not consider, this formula tells us that there are 673 vertices. This is the first time the true answer disagrees with the incorrect formula $(n+2)! + 1$ (which gives 721).

This story has a happy ending, of sorts. Future mathematicians who stumble upon this problem will not have the experience I did, because the correct answer now also shows up in the OEIS: as sequence A380996 [81].

21.6 Technical details

This section contains all the geometric arguments that we will need, in this part of the textbook, to confirm our intuition about how planar graphs can, and cannot be drawn. Though I will not cite specific claims, I have made sure that every geometric fact we need is contained somewhere in this section.

First of all, we should be more precise about what a plane embedding exactly is. There is not much to say about vertices: every vertex is drawn as a point in the plane. An edge should be drawn as some path between its two endpoints; two edges do not contain any common interior points, and in particular their interiors do not pass through any vertices. But what kind of paths should we allow?

In principle, any kind of continuous simple curve will do: the image of a continuous injective function $f\colon [0,1] \to \mathbb{R}^2$, where the points $f(0)$ and $f(1)$ are the two endpoints of the edge. However, studying such things will drown us in a sea of topology for no practical benefit.

It is sufficient to consider edges which are *polygonal curves*: made up of finitely many line segments joined end to end (and not intersecting otherwise). In a closed polygonal curve, or *polygon*, the last line segment ends where the first segment starts. Polygonal curves and polygons can approximate any crazier shape arbitrarily well; you won't be able to tell the difference. Practically speaking, when you encounter a planar graph (for example, derived from a map specified by latitude and longitude coordinates), it will already have edges in this form. Paths and cycles in a planar graph also become polygonal curves and polygons, respectively, in a plane embedding.

To define faces in a plane embedding, we want to be able to tell which points are in the same connected region of the plane (separated by the edges). Our test for this also uses polygonal curves: two points x and y are in the same face if there is a polygonal curve from x to y that does not intersect the plane embedding.

Question:	Why is this notion of connectedness practically useful to us?
Answer:	It tells us which new edges we could add to the plane embedding, because the edges are also polygonal curves.

There are a few more things to say about this notion of connectedness. From the point of view of theory, the most fundamental is the Jordan curve theorem, which states an obvious-seeming fact: every polygon separates the plane into exactly two connected regions, an inside and an outside, with the polygon as their boundary. (That is, points on the polygon are exactly the points arbitrarily close to both the inside and the outside.) The Jordan curve theorem holds for all continuous curves, not just polygons, but even in the polygonal case it is not easy to prove; I will skip it, to avoid too lengthy a geometric detour.

From the point of view of algorithms, it is not that interesting to know merely that every polygon has an inside and an outside; we would like to know what they are! A standard test for this is the ray casting algorithm: given a point x and a polygon P, draw an infinite ray (or half-line) starting at x. Then x is inside P if the ray crosses P an odd number of times, and outside P if the ray crosses P an even number of times.

The choice of ray doesn't matter, and in programming applications, it's often fine to take a horizontal or vertical ray, for simplicity. We do run into some technicalities if the ray passes through a corner of P or contains an entire line segment of P. It's possible to make sense of this situation: we say that if a segment of P has an endpoint on the ray, that segment crosses the ray if it lies to the left of the ray, but not if it lies to the right. But this is very technical, and it's easier to avoid it if possible.

We can get a more general test as a consequence of this algorithm:

Claim 21.5. *Let P be a polygon, and let points x and y be connected by a polygonal curve whose segments never start or end on P, and never intersect a segment of P in a sub-segment, only in a point.*

Then x and y are on the same side of P if and only if the polygonal curve connecting them crosses P an even number of times.

Proof. We can subdivide the $x - y$ polygonal curve into segments that each cross P at most once. If a segment crosses P once, its endpoints are on opposite sides: this is verified by the ray casting algorithm with a ray parallel to the segment, starting at either endpoint. So with every crossing, the polygonal curve switches to a different side of P.

Since there are only two sides, the polygonal curve ends on the same side as it started if and only if it switched sides an even number of times. □

We can now easily test which points lie in the same face, but do not have an easy definition of the boundary of a face. To obtain it, we need a geometric understanding of boundary walks.

Given a plane embedding, pick a value $\varepsilon > 0$ much smaller than every distance between a line segment and an endpoint of another end segment. Then surround every line segment by a thin rectangular loop: if the line segment has length ℓ, draw a $2\varepsilon \times (\ell + 2\varepsilon)$ rectangle with the line segment at its center. Orient each of these loops counterclockwise.

To find a boundary walk containing a particular segment, start going around its thin rectangular loop on either of the long sides. If this trajectory hits the loop around a different line segment (which only happens near the end of the segment), switch to following that loop. Keep going like this until this trajectory returns where it started (as it must, because we only have finitely many segments).

The result is a polygon that never crosses the plane embedding. It also stays within distance ε of a walk in the planar graph: it can only switch from following one edge to following another a vertex they share, because it never gets close enough to another edge anywhere else. We say that this walk is a boundary walk in the graph, and the polygon is the corresponding boundary walk in the plane embedding. Although boundary walks do not always represent cycles, we still consider two boundary walks to be the same if one is a shifted and/or reversed version of another.

The following is immediate from the definition.

- Each edge either lies on one boundary walk twice, or on two boundary walks, since each thin rectangular loop has two long sides.
- Each boundary walk in the plane embedding is contained entirely in one face, and we say it is a boundary walk of that face; a face can have multiple boundary walks.

Let xy be an edge of a planar graph G, and let C be the polygonal curve representing xy in a plane embedding of G. Many times, such as in the proof of Euler's formula, we want to understand edge xy by comparing the plane embedding of G and the plane embedding of $G - xy$ obtained by erasing C. In that second plane embedding, C stays entirely in some face F, because C does not cross any other edge. The following claim is a more careful formal proof of Lemma 21.2.

Claim 21.6. *Either $F - C$ is a single face in the plane embedding of G, xy lies on its boundary walk twice, and xy is a bridge of G; or $F - C$ is the union of two faces in the plane embedding of G, xy lies on each of their boundary walks once, and xy is not a bridge of G.*

Proof. Starting from either side of a segment of C, construct boundary walks using xy, letting B_1 and B_2 be the corresponding polygons in the plane embedding; possibly, $B_1 = B_2$. If $B_1 = B_2$, then follow this polygon from one side of C to a point 2ε away on the other side, and connect by cutting across C. The result is a polygon P that only crosses C, and only once. No edge of $G - xy$ crosses P, and x and y are on opposite sides (because C crosses P once), so there is no $x - y$ path in $G - xy$: xy is a bridge. If $B_1 \neq B_2$, then each boundary walk in G uses edge xy only once, so it also contains an $x - y$ path in $G - xy$, proving that xy is not a bridge.

Each point $z \in F - C$ can be joined to a point of C by a polygonal curve not crossing the plane embedding of $G - xy$; before it touches C, it must cross either B_1 or B_2. So all points of $F - C$

are in the face containing B_1 or the face containing B_2. (The exception is points within ε of C, which can reach B_1 or B_2 by taking a step of length at most ε away from C.) So there are at most 2 faces, and if $B_1 = B_2$, there is only one face.

Conversely, if $F - C$ is a single face in the plane embedding of G, then take a line segment of length 2ε from B_1 to B_2 that cuts across C; its endpoints are both in $F - C$, so this line segment can be completed to a polygon P that does not cross the plane embedding of G again. As before, no edge of $G - xy$ crosses P, and x and y are on opposite sides (because C crosses P once), so there is no $x - y$ path in $G - xy$: xy is a bridge. □

The machinery we've built so far is enough to prove Euler's formula. As a special case, consider a planar graph G with no cycles: a forest. With n vertices and k connected components, the forest has $n - k$ edges, by Theorem 10.1. Substituting this into Euler's formula, we learn that the forest only has one face.

With at least two faces, the picture changes:

Claim 21.7. *If a plane embedding of G has at least two faces, then some boundary walk of every face contains a cycle.*

Proof. Let F be any face; since it is not the only face, we can pick a point $p \in F$ and a point $q \notin F$. The line segment pq leaves F, so at some point it must cross an edge xy on a boundary walk of F to do so. Replace pq by a short sub-segment, if necessary, to ensure that pq only crosses xy once, and does not cross any other edge of the plane embedding.

In the plane embedding of $G - xy$, points p and q are part of the same face (since pq no longer crosses any edge). By Claim 21.6, the boundary walk of F using xy only uses it once: after going from x to y, it is a $y - x$ walk that does not use edge xy. That $y - x$ walk contains a $y - x$ path, which together with edge xy becomes a cycle. □

So far, we've been using the geometry of polygons to understand plane embeddings, but we can do this in reverse, as well.

Claim 21.8. *If p and q are two points on a polygon, then a polygonal curve that joins p and q divides one side of the polygon into two connected regions.*

Proof. Here, we think of the polygon plus the curve joining p and q as a plane embedding of a cycle graph with an extra edge (whose endpoints are drawn at p and q). That extra edge falls under the second case of Claim 21.6, proving the claim. □

Claim 21.9. *If p, q, r, and s are four points appearing on a polygon P in that order, then a polygonal curve joining p and r cannot be drawn on the same side of the polygon as a polygonal curve joining q and s without crossing it.*

Proof. Suppose such a diagram existed; then it would be an embedding of K_4, with vertices placed at p, q, r, and s. Such an embedding does exist: K_4 is a planar graph. But it cannot have the shape it needs to have here!

By Euler's formula, there are 4 faces total ($4 - 6 + 4 = 2$). One face of the plane embedding is the side P containing neither of the added curves; it has length 4. We know less about the others, but by Claim 21.7 each contains a cycle, so has length at least 3. The sum of the face lengths is at least $4 + 3 + 3 + 3 = 13$, but it must also be 12 by the face length formula, which is a contradiction. Therefore the diagram we started with cannot exist. □

There is one final claim that we will need to construct new plane embeddings of graphs:

- Going from a plane embedding of G to a plane embedding of $G \bullet e$ in Chapter 22;
- Finding the dual graph of a plane embedding in Chapter 23;
- Finding a plane embedding of the graph of adjacencies of a map in Chapter 24.

Claim 21.10. *Given a plane embedding with a face F, a point $p \in F$, and points $q_1, q_2, \ldots, q_k$ on the boundary of F, it is possible to draw polygonal curves from p to each of $q_1, \ldots, q_k$ that don't cross each other or the boundary of F (and therefore stay inside F).*

Proof. We will draw the polygonal curves one at a time. First, draw a polygonal curve from p to a point on the same boundary walk as q_1 (which is possible by definition of both points being in F). Then, follow that boundary walk until its closest approach to q_1, and end with a straight line segment.

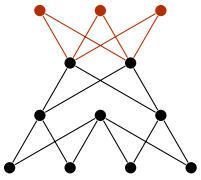
(It is useful to observe that that last line segment can never intersect any part of the plane embedding: it is entirely contained in one of the rectangular loops around a segment of an edge, which is too close to that segment to touch any other segments.)

Once this is done, we can think of the result as a bigger plane embedding where the point p and the polygonal curve we drew are part of the boundary of a face F' contained in F. We add another polygonal curve by taking a short step out to the point p' on the boundary walk nearest p, then drawing a polygonal curve from p' to the next q_i in the same manner as before.

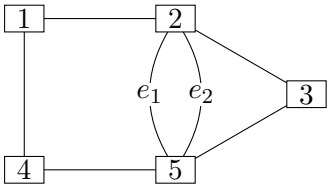
In general, this process can divide F' into two faces, and it's not guaranteed that the remaining points among $q_1, q_2, \ldots, q_k$ are all on the boundary of the same one. However, if this happens, then p is on the boundary of both of them, because the polygonal curve we drew from p is on the boundary of both, and we can repeat on each of the faces formed separately. □

21.7 Practice problems

1. Find a plane embedding of the *volcano graph*, shown below.



2. Consider the following planar multigraph:



Draw the following plane embeddings:

a) One where the outer face has length 3, with vertices $2, 3, 5$ and edge e_1 on its boundary, and the other faces have lengths 2, 4, and 5.

b) One where the outer face has length 4, with vertices $1, 2, 4, 5$ and e_1 on its boundary, and the other faces have lengths 3, 3, and 4.

c) One where the outer face has length 2, with edges e_1 and e_2 on its boundary.

3. Let G be a connected planar graph. Suppose that a plane embedding of G has r faces, all of length 4.

a) Use the face length formula to find the number of edges of G (in terms of r).

b) Use Euler's formula to find the number of vertices of G (in terms of r).

c) Can you find such a graph for $r = 8$? What about for $r = 10$?

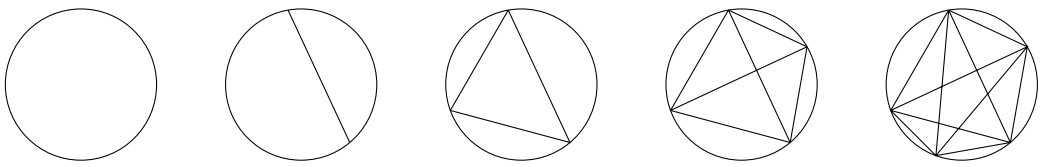
4. Use an argument similar to the proof of Proposition 21.1 to show that the complete graph K_5 is not planar. Start with a Hamilton cycle in K_5, then reason about where the remaining 5 edges can go.

5. Prove that all trees are planar graphs. Moreover, show that any tree has a plane embedding in which all the edges are straight lines.

(Hint: use induction. You'll need to show that if x is a leaf of T, then a plane embedding of $T - x$ can be extended to a plane embedding of T.)

6. Here is another deceptive problem whose real solution can be found with Euler's formula. (It is documented along with many deceptive patterns in "The Strong Law of Small Numbers" by Richard Guy [44].)

Put n points around a circle in random positions (not equally spaced) and draw all $\binom{n}{2}$ chords between those points; wiggle the n points around, if necessary, until no intersection point inside the circle lies on three or more chords.



In the examples above (where $n = 1, 2, 3, 4, 5$) the chords divide the circle into 1, 2, 4, 8, and 16 pieces, so you might be forgiven for thinking that the number for general n is 2^{n-1}; it's not! Use Euler's formula to find the correct answer.

7. a) Let G be a planar graph and let C be a cycle of length l in G. In every plane embedding of G, the cycle is a closed curve, but not necessarily the boundary of a face: its interior might be divided into several faces $F_1, F_2, \ldots, F_k$ by edges that are not part of C.

   Prove that in all such cases, the sum $\text{len}(F_1) + \text{len}(F_2) + \cdots + \text{len}(F_k)$ has the same parity as l: both are even, or both are odd.

   b) Use part (a) to prove that if G has a plane embedding in which every face has even length, then G is bipartite.

22 Planarity testing

The purpose of this chapter

This chapter presents three different approaches by which we can (sometimes or always) determine whether a graph is planar.

The first is Theorem 22.2, which is also very useful as a property of planar graphs: in Chapter 24, you will see how. It is especially important if you are not yet a very experienced mathematician to remember that this is very much not an if-and-only-if condition; do not use it in the wrong direction!

The second is Kuratowski's theorem and Wagner's theorem, which I group together, because they are very similar. As a planarity test, looking for minors is more powerful than looking for subdivisions, but minors are harder to understand—when teaching this material, I have often skipped Wagner's theorem for that reason. I include it here because in Chapter 24, it will be useful for us to know about edge contractions.

The third is the method of overlap graphs developed by Tutte. This is not as commonly studied, but I don't know why, because it's great. Any other equally systematic approach to planarity testing is much harder to learn.

22.1 Counting edges in planar graphs

In the previous chapter, we proved two tools that help us count vertices, edges, and faces in a plane embedding: the face length formula (Lemma 21.3) and Euler's formula (Theorem 21.4). We will now use these to prove a limit on the number of edges that a planar graph with n vertices can have.

For this, we will have to restrict our attention to simple graphs only, even though both results above apply to multigraphs as well. Otherwise, there is no possible bound we can prove!

Question:	Why can a planar multigraph with n vertices have arbitrarily many edges?
Answer:	Starting from an arbitrary plane embedding, we can draw in any number of loops at a vertex without crossings—imagine a flower with the loops as the petals, and the vertex at the center. If there is more than one vertex, we can also replace an edge by any number of parallel copies without affecting planarity.

We can engage in some meta-reasoning and ask: if our argument is to work for simple planar graphs, but not for planar multigraphs, what must it be doing? What sort of situations can only appear in the plane embedding of a multigraph?

Well, once again, we're back to the way that we can draw loops and parallel edges in a plane embedding. These are not unusual: they are the way we draw loops and parallel edges by default in a diagram of a multigraph. From the point of view of the counting tools we have, though, what makes them special is that they are boundaries of a face of length 1 (in the case of a loop) or 2 (in the case of two parallel edges).

Question:	Do loops and parallel edges have to be drawn as faces of length 1 or 2?
Answer:	No: at least in the case of some graphs, it's possible to draw the graph so that some part of it is inside the loop, or between the two parallel edges. (An example appears in the first practice problem at the end of the previous chapter.)

Question:	Are there any simple graphs with faces of those lengths?
Answer:	Just one: a graph with two vertices and a single edge between them has a plane embedding in which the outer face has length 2.

The following short lemma is the property we use to distinguish simple graphs from multigraphs. (From now on until we conclude our discussion of the number of edges in a planar graph, I will assume all graphs are simple.)

Lemma 22.1. *In a plane embedding of a simple graph with at least* 2 *edges, every face has length at least* 3*.*

Proof. If the plane embedding has multiple faces, then each face needs to be separated from the other faces somehow; its boundary must contain a closed curve, which corresponds to a cycle in the graph. The graph is simple, so a cycle in it must contain at least 3 edges.

If the plane embedding has only one face, then every edge of the graph must contribute 2 to the length of that face. There are at least 2 edges, so the face has length at least 4. □

When we feed Lemma 22.1 into the tools we have from the previous chapter and let it cook, we obtain the following theorem.

Theorem 22.2. *For all planar graphs with $n \geq 3$ vertices and m edges, $m \leq 3n - 6$.*

Proof. Choose a plane embedding of a graph with $n \geq 3$ vertices and m edges to work with for the rest of the proof. Let r be the number of faces it has; by Lemma 22.1, each face has length at least 3, so the sum of the lengths of all the faces is at least $3r$.

Question:	What if Lemma 22.1 does not apply, because $m < 2$?
Answer:	Then $m \leq 3n - 6$ automatically, because $3n - 6 \geq 3(3) - 6 = 3$.

By the face length formula, the sum of the lengths of all the faces is also equal to $2m$, giving us the inequality $2m \geq 3r$. We would like to combine this inequality with Euler's formula: $n - m + r - k = 1$. We do not know k, the number of connected components, but it's certainly at least 1, so we get a second inequality: $n - m + r \geq 2$.

Our goal in this proof is to establish a relationship between m and n (though it can be interesting to look at the other two pairs of variables, too), so we should eliminate r. We solve for r in the inequality $2m \geq 3r$ (getting $r \leq \frac{2}{3}m$) and in Euler's formula (getting $r \geq m - n + 2$). Putting them together, we get

$$m - n + 2 \leq \frac{2}{3}m$$

which we can rearrange to $\frac{1}{3}m \leq n - 2$, or $m \leq 3n - 6$. □

Question:	What would we learn if we eliminated m instead of eliminating r?
Answer:	From $m \leq n + r - 2$ and $m \geq \frac{3}{2}r$, then $n + r - 2 \geq \frac{3}{2}r$, or $r \leq 2n - 4$. This tells us the maximum number of faces in an n-vertex planar graph.

This chapter is called "Planarity testing", and Theorem 22.2 is our first planarity test: it lets us immediately conclude that some graphs are not planar! Here is an example:

Proposition 22.3. *The complete graph K_5 is not planar.*

Proof. We count: K_5 has $n = 5$ vertices and $m = 10$ edges. Since $3n - 6 = 9$, which is less than m, the conclusion of Theorem 22.2 does not hold. Therefore the hypotheses do not hold either, so K_5 is not a planar graph. □

Question:	If an n-vertex graph has $3n - 6$ or fewer edges, can we conclude from Theorem 22.2 that it is planar?
Answer:	No: the theorem gives a necessary condition for planarity, but not a sufficient one. For example, take K_5 and add 95 isolated vertices, and you'll get a 100-vertex graph with only 10 edges which is not planar.

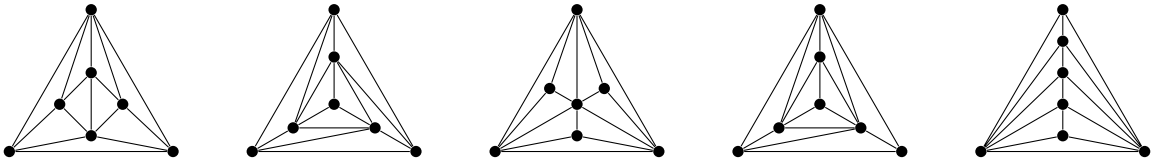


Figure 22.1: Triangulations with 7 vertices

22.2 Triangulations

Whenever we prove an inequality, a natural question to ask is: what can we say about the cases where equality holds? What kind of planar graphs have $m = 3n - 6$?

To draw conclusions about such graphs, we should look back at our proof, and look at every place where an inequality appeared. This is an extremely useful idea not just in graph theory, but in other areas of math!

1. We wrote Euler's formula as an inequality: $n - m + r \geq 2$. If it were the case that $n - m + r > 2$, we would conclude that $m < 3n - 6$.

   So if a planar graph with $n \geq 3$ vertices satisfies $m = 3n - 6$, then $n - m + r$ must be equal to 2, meaning that the graph is connected.

2. The inequality $m \leq 3n - 6$ combines Euler's formula with the inequality $2m \geq 3r$. If we had started with the strict inequality $2m > 3r$, we would have arrived at the strict inequality $m < 3n - 6$, instead.

   So if a planar graph with $n \geq 3$ vertices satisfies $m = 3n - 6$, it must satisfy $2m = 3r$.

3. The inequality $2m \geq 3r$ itself comes from another inequality, but one we only stated in words: Lemma 22.1, which says that every face has length at least 3. (For all faces F, $\text{len}(F) \geq 3$.)

   So if a planar graph with $n \geq 3$ vertices satisfies $m = 3n - 6$, then every face must have length exactly 3, and this must be true in every plane embedding of the planar graph.

Now that we've understood these extremal examples, we give them a name, based on the fact that all of their faces are (in some sense) triangles. We call a plane embedding of a connected graph in which all faces have length 3 a **triangulation**. For each n, there are many n-vertex triangulations; see Figure 22.1 for some examples when $n = 7$. (These are all the 7-vertex possibilities, up to isomorphism of the planar graph. I know this because I first looked up sequence A000109 in the OEIS [79] to confirm there are 5 of them, then searched the House of Graphs [20] to find out what they are.)

To be clear, "triangulation" is a term for the plane embedding, not for the planar graph. How do we refer to the planar graph, then? The corresponding notion is a *maximal planar graph*: a planar graph that stops being planar if any edge (that it does not already have) is added to it. But to avoid sneaking in a claim that has not been justified, we need the following proposition.

Proposition 22.4. *For a planar graph G with $n \geq 3$ vertices, the following are equivalent:*

(i) G has $3n - 6$ edges.

(ii) Every plane embedding of G is a triangulation.

(iii) G is a maximal planar graph.

Proof. We have already proven that (i) $\iff$ (ii). G has exactly $3n-6$ edges if and only if every inequality in the proof of Theorem 22.2 is an equality: if and only if every face has length exactly 3. This has to happen regardless of the plane embedding we choose at the beginning of that proof, so it must be true for every plane embedding.

Question:	Which other implication between (i), (ii), and (iii) is easiest to show?
Answer:	The implication (i) $\implies$ (iii) also follows from Theorem 22.2. If G has $3n-6$ edges, and we add a new edge to G, then we get a graph with $3n-5$ edges, so by the contrapositive of Theorem 22.2, the new graph is not planar.

The part of Proposition 22.4 that we still need to prove is that (iii) also implies (i) and (ii). In other words, there are no maximal planar graphs that "get stuck" before reaching $3n-6$ edges. We will show that (iii) implies (ii), and we will do it by showing the contrapositive: if G has an embedding that's not a triangulation, then G is not a maximal planar graph.

Our strategy has a short description: in a plane embedding of G that isn't a triangulation, we find a face F with $\text{len}(F) \geq 4$, and then we add an edge between two vertices on the boundary of F, drawing it inside F. The reason this is not the entire proof is that we have to make sure the edges do not already exist inside F.

In order for that to even be a problem, F must not be the only face, which means that there is a cycle on the boundary of F separating it from the other faces. This is either a cycle of length 4 or more, or else a cycle of length 3 with some more vertices of F "inside" the cycle; in either case, there are at least 4 vertices on the boundary of F.

If there are 5 or more vertices on the boundary of F, then they cannot all be adjacent: G would then contain a copy of K_5, which we know cannot be drawn in the plane. If there are only 4 vertices, and they are all adjacent, then the plane embedding of G would contain a plane embedding of K_4 in which F is a face. This is impossible: by (i) $\implies$ (ii), every plane embedding of K_4 is a triangulation, and cannot contain a face like F of length more than 3.

Question:	Does anything change if F is the outer face?
Answer:	Not substantially; the outer face is outside the cycles on its boundary, rather than inside them, but nothing changes aside from that one word.

In all cases, at least two vertices x, y on the boundary of F are not already adjacent in G. If we add edge xy to G, the result is still planar, because drawing a curve from x to y inside F gives a plane embedding of $G+xy$. This completes the proof: it shows that G is not a maximal planar graph. □

22.3 Girth and planarity

Before we go on to necessary and sufficient conditions for planarity, let's try to squeeze a bit more power out of the approach used to prove Theorem 22.2, because counting edges is a simpler test than just about anything else we could try.

The most common scenario is when G is a bipartite planar graph. In this case, the boundary of a face cannot be a cycle of length 3: by Theorem 13.1, bipartite graphs cannot contain such cycles! This allows us to sharpen our inequality.

Question:	What if the boundary of a face is not a cycle?
Answer:	In general, the length of the face is the total length of its boundary walks, and in a bipartite graph, all closed walks have even length.

If the minimum length of a face is 4, then we replace the inequality $2m \geq 3r$ in the proof of Theorem 22.2 by the inequality $2m \geq 4r$. As before, when we want an upper bound on m, we may assume G is connected, which lets us apply Euler's formula: $n - m + r = 2$. Eliminating r and simplifying, we conclude:

Theorem 22.5. *For all planar bipartite graphs with $n \geq 3$ vertices and m edges, $m \leq 2n - 4$.*

To appreciate the power of this approach, we can go back to Proposition 21.1 from the previous chapter. At the time, we need a technical argument that looked closely at the geometry of a plane embedding in order to prove that $K_{3,3}$ is not planar. With Theorem 22.5, the proof is simple: $K_{3,3}$ is a bipartite graph with $n = 6$ vertices and $m = 9$ edges. $9 > 2 \cdot 6 - 4$, so $K_{3,3}$ is not planar.

This argument is just one case of an even more general approach. Define the *girth* of a graph G to be the length of the shortest cycle in G. (This is always at least 3, and in bipartite graphs it is always at least 4.) In a graph with no cycles at all (a forest), the girth is sometimes defined to be ∞, but that will not matter in this chapter. In any case, we don't need a planarity test for forests: they are always planar.

Theorem 22.6. *Let G be a planar graph with at least one cycle.*

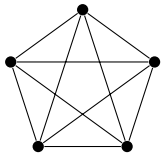
If G has $n \geq 3$ vertices, m edges, and girth g, then $m \leq \frac{g}{g-2}(n-2)$.

Proof. Fix a plane embedding of G. It will have at least two faces, because a cycle separates the plane into two regions. Conversely, when there is more than one face, every face has a cycle in its boundary. Therefore the girth g is also a lower bound on the length of the faces of the plane embedding.

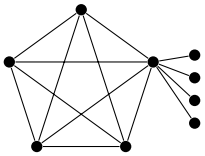
Now we can continue as before. Let the plane embedding have r faces; then the sum of the lengths of the faces (which is $2m$, by the face length formula) is at least rg. Combining the inequality $2m \geq rg$ with the equation $n - m + r = 2$ from Euler's formula, we get

$$2 - n + m = r \leq \frac{2m}{g}.$$

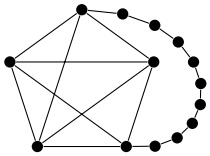
This simplifies to $m \leq \frac{g}{g-2}(n-2)$, the inequality we wanted. □



(a) K_5 is not planar



(b) An unsatisfying extension



(c) A subdivision of K_5

Figure 22.2: Non-planar graphs based on K_5

22.4 Subdivisions

Thus far, the graphs $K_{3,3}$ and K_5 have not been planar because they have too many edges. It is easy, but not very exciting, to find more graphs that are not planar, simply because they have a copy of $K_{3,3}$ or K_5 inside them. Figure 22.2b shows an example of this. We've added several vertices and edges to K_5; the result is even connected. However, the extra vertices and edges aren't really contributing anything to the non-planarity; the only reason that the resulting graph is not planar is because it contains a copy of K_5 inside it.

But we can consider stranger things than just subgraphs. Take a look at the graph in Figure 22.2c, for example. This graph does not have K_5 as a subgraph: it has 5 vertices in all the same places as Figure 22.2a, but one of the edges is missing. In place of that long edge is a path through some entirely new vertices.

Question:	Why is the graph in Figure 22.2c not planar?
Answer:	If we could draw it in the plane, we could simply erase the dots on the long path, and get a drawing of K_5.

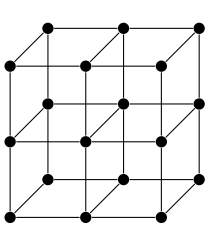
We can generalize this construction. We say that to **subdivide** an edge xy in a graph G means to create a new vertex z and replace edge xy by edges xz and yz. (In a diagram, this corresponds to drawing a dot representing z in the middle of edge xy.)

A **subdivision** of a graph G is a graph that can be obtained from G by subdividing edges zero or more times. (We consider G itself to be a subdivision of G.)

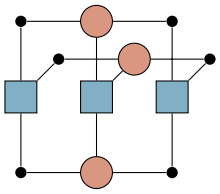
Question:	Motivated by Figure 22.2c and our argument for it, what is the relationship between subdivisions and planarity?
Answer:	If H is a subdivision of G, then G is planar if and only if H is planar: subdividing edges does not change planarity.

There is a reason why we have focused on two specific graphs that are not planar: the graphs K_5 and $K_{3,3}$. That reason is the following theorem, proved in 1930 by Casimir Kuratowski [67]:

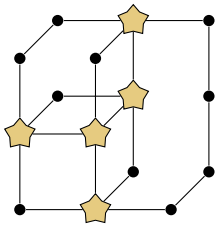
Theorem 22.7 (Kuratowski's theorem)**.** *A graph G is planar if and only if it G contains a subdivision of K_5 or $K_{3,3}$.*



(a) A non-planar example



(b) A $K_{3,3}$ subdivision



(c) A K_5 subdivision

Figure 22.3: Showing that a graph is not planar using Kuratowski's theorem

Thus, subdivisions of K_5 and $K_{3,3}$ are the only reasons why a graph might fail to be planar. (Strictly speaking, I should write, "G contains a copy of a subdivision of K_5 or $K_{3,3}$, but nobody is that strict about it, so I won't be, either.)

Kuratowski's theorem is another example of a theorem that lets us concisely identify a reason why a problem in graph theory might have no solution. This is not necessarily related to how hard a problem is; it might be quite hard to find a plane embedding of a large graph, and it might also be hard to find a subdivision of K_5 or $K_{3,3}$ inside it. However, once you have a plane embedding, you can show it to anyone else, and instantly (or very quickly) convince them that a graph is planar. Working directly from the definition, there is not a lot you can show someone to quickly convince them that a graph is not planar.

That's exactly what Kuratowski's theorem provides. The subdivisions of K_5 and $K_{3,3}$ are obstacles to planarity. But Kuratowski's theorem mostly isn't about telling us that they're obstacles: we already knew that part. If we believe that K_5 and $K_{3,3}$ are not planar, then we can know that their subdivisions are not planar just by an argument like the one we used for Figure 22.2c. No, Kuratowski's theorem is about guarantees: it tells us that these obstacles are the only ones we need to worry about.

I will not give you a proof of Kuratowski's theorem. However, I will show you how we can look for a subdivision of K_5 or $K_{3,3}$ in a graph that is not too large. (If the graph is very large, then the task should be delegated to a computer, but it helps to have some understanding of what the computer could be trying to do.)

The example I will use is the graph in Figure 22.3a, which I will refer to as G for the rest of this section. A subdivision of $K_{3,3}$ inside G is shown in Figure 22.3b, and a subdivision of K_5 in Figure 22.3c. But how do we find them?

Question:	Must both kinds of subdivisions exist in G, if it is not planar?
Answer:	Not necessarily; Kuratowski's theorem only promises one of the two kinds of subdivisions. Even if both exist, we don't need to find both!

A first step might be to decide whether it is a subdivision of $K_{3,3}$ or a subdivision of K_5 that we want. If it exists, the subdivision of K_5 might be easier to look for, because K_5 has fewer vertices. However, K_5 has five vertices of degree 4, and this remains true for every subdivision of K_5. If we're testing a graph G with fewer vertices of degree 4 or more, we can give up on finding a subdivision of K_5, and focus on $K_{3,3}$. But G has enough degree-4 vertices (and even two degree-5 vertices) to find both.

In either case, we should begin by deciding which vertices will be the "key" vertices of the subdivision: the vertices of degree 3 or 4, as opposed to the vertices of degree 2 in the middle of subdivided edges. It is a good idea to pick central and high-degree vertices of the graph, to make it easier to find the paths later. A natural first choice is one of the two "center" vertices of G. (Call these vertices x and y, for future reference.)

If we did not know whether G is planar, we might also spend some time trying to find a plane embedding of G. This can also tell us something, even if we fail! For example, after trying for a while, you might be able to find a plane embedding of almost all of G, which is missing the edge xy between the two center vertices.

Question:	What would such an embedding tell us?
Answer:	Since $G - xy$ is planar, it does not contain a subdivision of $K_{3,3}$ or K_5. Therefore whatever subdivision we hope to find in G must use edge xy somehow.

It is not, logically speaking, necessary for vertices x and y to be two of the key vertices of the subdivision. But it already seemed like a good idea to use these vertices before, so we might as well start with that theory.

It might take some trial and error to place the remaining key vertices. It is often a good idea to place as many of them close by as possible, so that they can be connected directly by edges; then, only a few long paths are necessary to draw the remaining connections.

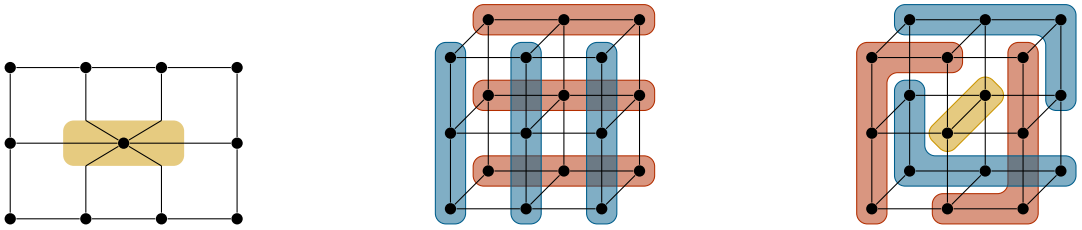
It can help to switch between different ways of thinking about the graphs we're subdividing—especially $K_{3,3}$, where we can either imagine the bipartition with three vertices on each side, or the hexagon with three chords. For a subdivision of K_5, we can break down the process of finding it into stages:

1. Choose x and y to be our first two vertices.
2. Use edge xy as an edge of the subdivision, but also find three more longer $x - y$ paths, sharing no other vertices.
3. From each of those $x - y$ paths, choose a vertex to be a key vertex of the subdivision. Now, find a cycle through those three vertices, using no other the vertices used in previous stages.

A similar breakdown into stages could also work in other examples.

22.5 Graph minors

Another useful operation that preserves the planarity of a graph is edge contraction. If xy is an edge of graph G, the graph $G \bullet xy$ (also sometimes denoted G/xy) is the graph obtained from G by deleting vertices x and y and adding a new vertex z adjacent to every vertex which, in G, was adjacent to either x or y. The operation of going from G to $G \bullet xy$ is called **contracting** edge xy.



(a) Contracting an edge (b) A $K_{3,3}$ minor (c) A K_5 minor

Figure 22.4: Examples of edge contractions and minors

In some applications, it makes sense to define $G \bullet xy$ as a multigraph. In that case, for every edge between a vertex w and either x or y in G, there is an edge between w and z in $G \bullet xy$; if there are multiple edges between x and y, and only one of them is contacted, the remaining edges become loops at z. In our case, loops and parallel edges are not relevant to track, so we will keep $G \bullet xy$ a simple graph.

Intuitively, given a plane embedding of G, we obtain a plane embedding of $G \bullet xy$ by first erasing everything within a small distance of edge xy (including the endpoints x and y). Within the erased region, draw vertex z, and change the trajectory of every edge formerly incident to x or y, so that instead it heads to z once it enters the erased region. Figure 22.4a shows an example of this intuitive idea, contracting an edge in a 3×4 grid graph.

Contracting edges is almost, but not quite, the reverse operation of subdividing edges. If we subdivide an edge xy (creating a new vertex z) and then contract edge xz (calling the combined vertex x) then we obtain the original graph again. However, contracting edges can do more than just undo edge subdivisions, when both endpoints of the contracted edge have degree 3 or more.

We say that a graph H is a *minor* of another graph G if it is a subgraph of a graph obtained from G by contracting edges zero or more times. If G is planar, then every minor of G is also planar, because contracting edges will not affect planarity, and neither will going from G to a subgraph.

Rather than think of a minor H as the result of a sequence of operations done to G, it can help to trace back where every vertex of H comes from.

Question:	What is the simplest "origin story" of a vertex of H?
Answer:	It could have been a vertex of G to begin with.

Question:	What is next simplest?
Answer:	A vertex of H could have started as an edge of G.

Question:	What else could have happened?
Answer:	A vertex of H could be the result of contracting an edge whose endpoints were themselves the results of edge contraction(s). We could have an arbitrarily large set of vertices in G that all collapse down to one vertex in H.

There is only one restriction on the set of vertices that collapse down to one vertex in H. Such a set only shrinks due to contracting an edge between two of its vertices, so if we can get it down to a single vertex, it's because the set was originally a connected subgraph of G. So, we can identify the vertices of H with disjoint connected subgraphs of G; if two vertices of H are adjacent, then the corresponding subgraphs of G must have at least one edge between them.

Figure 22.4b and Figure 22.4c show two examples of this. Well, to be precise, they show only the subgraphs which are to be contracted to vertices. In Figure 22.4b, the three vertical (blue) subgraphs become one side of $K_{3,3}$, and the three horizontal (red) subgraphs become the other side. To verify that we really do get a $K_{3,3}$ minor, we need to check that between each red vertex and each blue vertex, there is at least one edge. Verifying a K_5 minor takes less explanation: in Figure 22.4c, we need to check that there is an edge between every pair of the 5 circled regions.

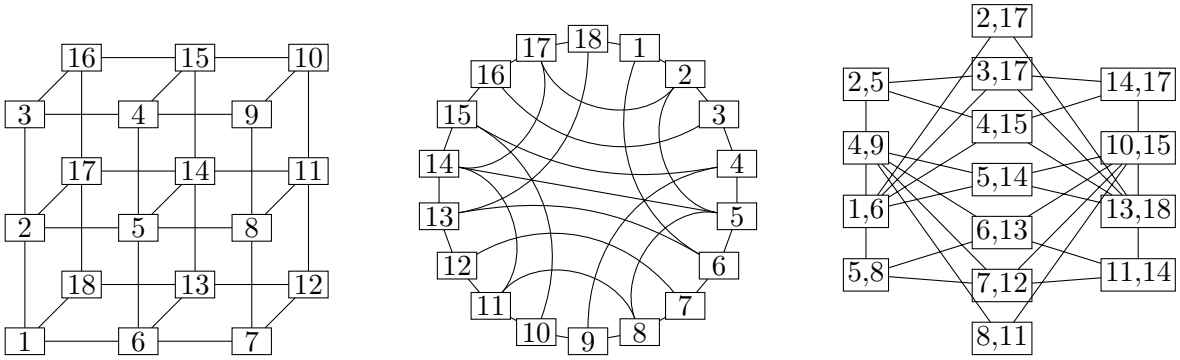
Question:	Why is finding a $K_{3,3}$ minor or a K_5 minor in a graph G interesting?
Answer:	It shows that G is not planar: if G were planar, then all its minors would be planar, which is not true of $K_{3,3}$ or K_5.

Klaus Wagner was the first to study graph minors in 1937 [103]. Among other results, he proved the analog of Kuratowski's theorem for graph minors:

Theorem 22.8 (Wagner's theorem)**.** *A graph G is planar if and only if it does not have a minor isomorphic to $K_{3,3}$ or K_5.*

As a characterization of planar graphs, Wagner's theorem follows from Kuratowski's theorem: if G contains a subdivision of $K_{3,3}$ or K_5, then by contracting all the edges along paths in that subdivision, we can obtain a $K_{3,3}$ or K_5 minor. As far as we're concerned in this book, Wagner's more significant contribution is the concept of graph minors. Not all minors arise from subdivisions; in fact, it's possible to find a minor isomorphic to H in a graph which does not contain any subdivisions of H. So looking for a $K_{3,3}$ minor or a K_5 minor is easier than looking for a subdivision—once you're comfortable with the definition of a minor.

Outside the study of planar graphs, Wagner's work proved to be a greater influence on graph theory, because classifying graphs using their minors turned out to be a much more useful approach than classifying graphs using their subdivisions.



(a) A vertex labeling (b) A Hamilton cycle (c) Its overlap graph

Figure 22.5: Showing that a graph is not planar using overlap graphs

22.6 Overlap graphs

A different approach to testing whether a graph is planar was first described in 1959 by Tutte [98]. It is less well-known than Kuratowski's theorem, but in my opinion, it is easier to use in small examples, and it can be the basis for efficient algorithms for planarity testing, as well. What's more, it is not just a way to prove that a graph is not planar; it can help find a plane embedding.

The idea is similar to the approach we took in the previous chapter to prove that $K_{3,3}$ is not planar. It begins by choosing a cycle in the graph we want to test for planarity. A Hamilton cycle works best, but of course Hamilton cycles can be hard to find, and we don't want to turn an easier problem into a harder problem! We can try the test with any cycle; however, the longer it is, the better.

As an example, I will show you how this approach works on the graph G in Figure 22.3. I have drawn it again with the vertices labeled in Figure 22.5a for two reasons: for ease of use, and also to point out a Hamilton cycle. In Figure 22.5b, that Hamilton cycle is drawn around a circle, and this picture is good to keep in mind for intuition.

In a hypothetical plane embedding of G, this cycle (and any other cycle) must appear as a closed loop of some sort. The rest of G must be drawn somewhere either inside that closed loop or outside it. In this example, "the rest of G" is just the edges which are not part of the cycle.

In general, given the graph G and a cycle C, let S be the set of vertices on C. We gather the vertices and edges of G which are not on C into *fragments*, defined as follows:

- Every edge of G whose endpoints are both in S is a fragment.
- Each connected component of $G - S$, together with the edges it has to S, is a fragment.

The motivation behind this definition is that a fragment is something that must either be drawn entirely inside C in a plane embedding of G, or else entirely outside it: it cannot cross C. For two different fragments, on the other hand, we can make different decisions. In our example, all fragments are edges, because C is a Hamilton cycle.

Question:	Why shouldn't we have defined edges like $\{2, 5\}$ and $\{2, 17\}$, which share an endpoint, to be part of the one fragment?
Answer:	Because we don't have to draw them on the same side of the cycle: to draw edge $\{2, 5\}$ inside the cycle and edge $\{2, 17\}$ outside it, no edges need to cross.

The decisions that we make for different fragments are not entirely independent. For example, edge $\{1, 6\}$ and edge $\{2, 17\}$ would intersect if we drew them both inside the Hamilton cycle, or both outside. To keep track of this information, we define a new graph, called the *overlap graph* of the cycle C. Its vertices are fragments, and they are adjacent if they "overlap": if they cannot be drawn on the same side of C.

A practical definition of the overlap graph in our case is that it has a vertex for each edge not part of C; two vertices are adjacent if, in Figure 22.5b, they cross. This remains a perfectly good visual rule in general, but we'd like a combinatorial rule, because a computer cannot look at a drawing and see if two edges cross. The combinatorial rule for when two fragments overlap is this:

- There are four vertices w, x, y, z appearing in that cyclic order around the cycle C.
- One fragment includes an edge to vertex w and an edge to vertex y; this may be satisfied by the fragment being edge wy.
- The other fragment includes an edge to vertex x and an edge to vertex z; this may be satisfied by the fragment being edge xz.

The overlap graph in our example is shown in Figure 22.5c.

Guided by the overlap graph, we must choose for each fragment whether to draw it inside C or outside C. The rule is that if two fragment are adjacent in the overlap graph, then one of them must be inside C, and the other must be outside C.

Question:	In terms of the overlap graph, what are we trying to find?
Answer:	A 2-coloring, or bipartition! The fragments we decide to draw inside C are one side of the bipartition, and the fragments we decide to draw outside C are the other side.

We know from Chapter 13 that we can efficiently test if the overlap graph is bipartite, and Theorem 13.1 tells us that the overlap graph is either bipartite, or contains a cycle of odd length. That cycle is our proof that the graph cannot be drawn in the plane!

Question:	Is there a cycle of odd length in the overlap graph in Figure 22.5c?
Answer:	Yes: for example, the fragments $\{1, 6\}$, $\{4, 9\}$, and $\{5, 14\}$ form such a cycle.

Question:	If the overlap graph were bipartite, would that be enough to conclude that G is planar?
Answer:	Not in general: it's also possible that a single complicated fragment cannot be drawn together with C without crossing, regardless of what other fragments are doing. The overlap graph will not detect this.

However, in an example like this one, where C is a Hamilton cycle, deciding which fragments are inside and which fragments are outside C is all that there is to finding a plane embedding. A bipartition of the overlap graph tells us exactly how to draw G.

If the overlap graph is not bipartite, then it can also guide us to finding a subdivision of K_5 or $K_{3,3}$. The very simplest case is what we found in this example: three edges $\{1, 6\}$, $\{4, 9\}$, and $\{5, 14\}$ which all overlap. In that case, the cycle C together with those three edges is a subdivision of $K_{3,3}$.

In more complicated cases, more work needs to be done, but the overlap graph can still simplify the search. Take the cycle C, and the fragments forming an odd cycle in the overlap graph: just this portion of the graph alone is not planar, so it is all that is necessary to find a subdivision of K_5 or $K_{3,3}$.

22.7 Practice problems

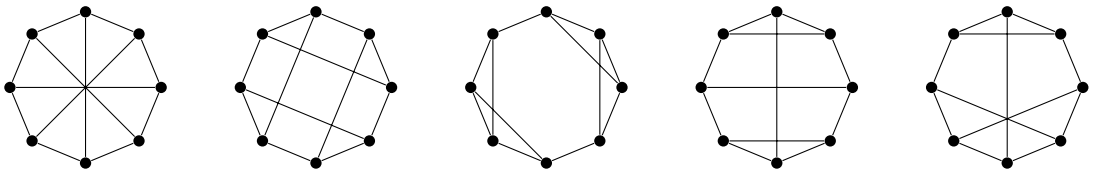
1. The two graphs below were used as an example in the previous chapter.

   One of these is planar, and the other one is not.

   a) Identify the planar graph, and draw a plane embedding.

   b) Using one of the tools in this chapter, prove that the other graph is not planar.

2. The five connected 3-regular graphs with 8 vertices are all shown below. Determine which of them are planar, and which are not.



3. Prove that the Petersen graph is not planar in four different ways:

   a) By using Theorem 22.6.

   b) By finding a subdivision isomorphic to $K_{3,3}$.

   c) By finding a minor isomorphic to K_5.

d) By finding a long cycle in the Petersen graph for which the overlap graph is not bipartite.

4. What is the maximum number of edges in an n-vertex planar graph if we know it has a plane embedding with two faces of length 6?

5. a) Let G be a connected graph with n vertices and $n+2$ edges. Prove that G is planar.

   b) Let G be a graph with n vertices and $n+3$ edges obtained by starting with the cycle graph C_n and adding 3 more edges.

   When is G planar, and when is G not planar?

6. The graph in Figure 22.3a is a $1\times2\times2$ three-dimensional grid graph. In general, the $a\times b\times c$ grid graph has vertices which are 3-dimensional points (x_1, x_2, x_3) with $x_1 \in \{1, 2, \ldots, a\}$, $x_2 \in \{1, 2, \ldots, b\}$, and $x_3 \in \{1, 2, \ldots, c\}$; two vertices are adjacent if they are at distance 1 from each other. Thinking of Figure 22.3a as a 3-dimensional drawing can give you an idea of what an $a \times b \times c$ grid graph looks like.

   For which a, b, and c is the $a \times b \times c$ grid graph planar?

7. (Putnam 2007) Let a triangulated polygon be a plane embedding in which every face except the outer face must have length 3. Prove that there is a function $f(n)$ such that if the outer face of a triangulated polygon has length n, and every vertex not on the boundary of the outer face has degree at least 6, then the triangulated polygon has at most $f(n)$ faces.

23 Polyhedra

The purpose of this chapter

To a graph theorist, Euler's formula is a theorem about planar graphs. To almost every other mathematician, it is a theorem about three-dimensional solids. In this chapter, we'll see the connection, and put graph theory to work in understanding 3D geometry.

In the middle of this chapter is a section on dual graphs, which I've included here because the duality between Platonic solids is a particularly striking instance of dual graphs at work.

The discharging method used to prove Theorem 23.3 is a common way to use Euler's formula in proofs; it is also not a proof strategy that's easy to come up with on your own. So I think it's a particularly valuable proof to study, in case you ever encounter a problem in which this method can be used.

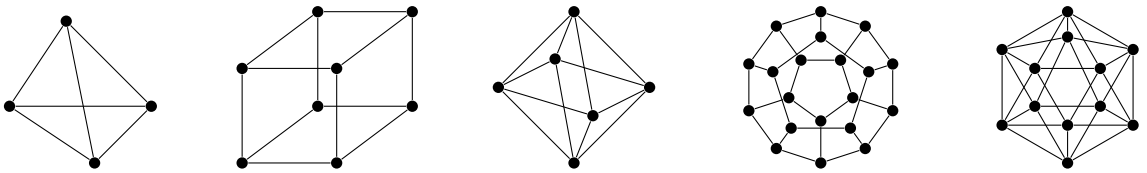
23.1 The Platonic solids

Regular polygons in the plane are abundant. For every $n \geq 3$, it is possible to draw a polygon with n equal sides and n equal angles. The sides can be however long we want them to be, but the angles must all have measure $\frac{n-2}{n} \cdot 180°$ (or $\frac{n-2}{n} \cdot \pi$, in radians). The reason for this is that we can divide a regular n-sided polygon (or n-gon) into $n-2$ triangles by drawing lines between its corners: two ways to do this are shown in Figure 23.1. In a triangle, the sum of angles is always 180°; adding up the angles of all triangles gives $(n-2) \cdot 180°$, but this is also equal to the sum of all n angles of the regular n-gon.

Things change when we go up to 3 dimensions. The 3-dimensional version of a polygon is called a **polyhedron**. (The plural is "polyhedra".) This is a shape with flat polygonal faces which come together at their sides and at their corners. If we want a polyhedron to be as regular as possible, then we can ask for the faces to be congruent regular polygons, with an equal number of them meeting at every corner.

(a) A regular 7-gon (b) 5 triangles in a 7-gon (c) A regular 8-gon (d) 6 triangles in an 8-gon

Figure 23.1: Dividing regular polygons into triangles



(a) Tetrahedron (b) Cube (c) Octahedron (d) Dodecahedron (e) Icosahedron

Figure 23.2: The Platonic solids

This much regularity is a lot to ask for! In fact, there are only five possibilities once we ask for this much, shown in Figure 23.2. They are known as the *Platonic solids*, named after their description in Plato's *Timaeus*, where Plato connects four of them to the four classical elements. They were analyzed more geometrically in Euclid's *Elements*, and have been studied since then both mathematically and mystically [37].

Question:	How many sides do the Platonic solids have?
Answer:	Going from left to right in Figure 23.2: 4, 6, 8, 12, and 20. The Greek names of the solids refer to the number of sides: "tetra-" is a prefix that means 4, "octa-" means 8, and so on.

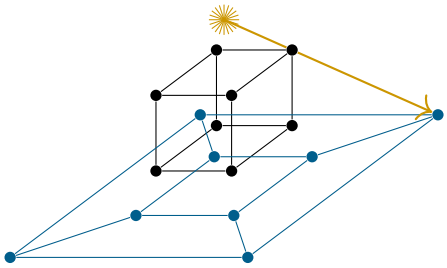
We will try to understand why there are just five of these solids. For this, it is necessary to connect the problem to graph theory somehow. You can probably already guess how we do it, by looking at the diagrams in Figure 23.2, and maybe recognizing the cube (Figure 23.2b) and dodecahedron (Figure 23.2d) from previous chapters. Each Platonic solid has an associated graph called its *skeleton graph*. Its vertices are the corners of the polyhedron, and its edges are the line segments where two of the polygonal faces meet.

(Actually, these objects are also called "vertices" and "edges" by geometers studying polyhedra. This is not a coincidence: graph theory gets its terminology from geometry, and not the other way around!)

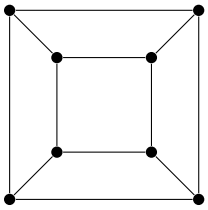
However, the skeleton graphs of the Platonic solids are not just any kind of graph. They are planar graphs, and they have a plane embedding in which the faces (the polygonal sides of the polyhedron) become the faces of the plane embedding. This allows us to use the theory of planar graphs we have developed in the previous two chapters to solve a geometric problem.

There are two ways to get an intuition for how a polyhedron can be turned into a planar graph. One way is to imagine the polyhedron to be made of rubber; then, inflate the polyhedron until it becomes spherical, bending the edges into arcs on the sphere. (You should imagine the edges to be painted on, so that they do not get completely forgotten in this process.) Then, poke a hole in the rubber sphere, and stretch it out until it is flat; the drawing of the polyhedron on the sphere turns into a drawing in the plane.

Question:	Is this possible for any polyhedron you can imagine?
Answer:	No: some polyhedra have "holes" in them, and will inflate to a shape that is not a sphere. All five of the Platonic solids do inflate to spheres.



(a) The shadow of a cube



(b) A plane embedding of the cube

Figure 23.3: Going from a polyhedron to a planar graph

For some reason, mathematicians often feel that arguments involving imaginary rubber spheres are insufficiently rigorous. So here is another approach that is both visual and can be made rigorous. Hold the polyhedron above a horizontal plane, oriented so that a face on top is parallel to this plane. Place a bright light slightly above that face, close enough so that an observer at the bright light would see the top face and nothing else. Then the vertices and edges of the polyhedron will cast a shadow onto the horizontal plane, and that shadow will exactly be a plane embedding of the skeleton graph. An example is shown in Figure 23.3a.

The reason I say this can be made rigorous is that the projection via bright light can be described mathematically: for example, if the bright light is at point $(0, 0, 1)$ in $\mathbb{R}^3$, and the horizontal plane is the plane $z = 0$, then the shadow of a point (x, y, z) has coordinates $(\frac{x}{1-z}, \frac{y}{1-z}, 0)$. Geometrically, the bright light, a point on the polyhedron, and its shadow are collinear.

We still need to take care to make sure that no edges cross in the shadow. A sufficient condition for this is to start with a *convex polyhedron*: one with the property that if two points A, B are contained in the polyhedron, then so is the entire line segment $\overline{AB}$. When a ray of light shines on a convex polyhedron, it always enters the interior of the polyhedron once (through the top face) and exits once. For two edges to cross in the shadow, the ray of light pointing at that crossing would need to enter and exit multiple times: once in the top face, and once for each edge that intersects at the crossing.

All of the polyhedra we consider in this chapter will be convex, and so all of them can be described as planar graphs—not just the Platonic solids. This correspondence can be taken even further, though. In general, a graph has a plane embedding exactly when it has a spherical embedding: when it can be drawn on the surface of a sphere without crossing. The reason is that we can take a spherical embedding and turn it into a plane embedding in exactly the same way that we've just done it for a polyhedron. Going the other way, a plane embedding can be drawn on the sphere just by using a small portion of the sphere that looks basically flat. We do this (with a certain very large sphere) every time we draw a picture in the dirt with a stick.

23.2 Classifying the Platonic solids

In two dimensions, there are infinitely many regular polygons. So why are there only five Platonic solids in three dimensions? This is, in part, something we can prove from Euler's formula, with a quibble I will mention at the end of this section.

We can describe a Platonic solid by a pair (p, q) where every face has p sides, and q faces meet at every vertex. Geometrically, we must have $p \geq 3$ and $q \geq 3$.

Question:	Why $p \geq 3$?
Answer:	A polygon cannot have fewer than 3 sides.

Question:	Why $q \geq 3$?
Answer:	If we try to have only two polygons meet at a vertex, they end up lying flat against each other. I suppose I can't stop you from declaring that gluing two regular n-gons back-to-back is a Platonic solid with 2 faces at every vertex, but Plato wouldn't have been on board with this.

We can further narrow down the options for p and q by using the properties of a planar graph. (The planar graph must be connected, because a Platonic solid should be connected: two cubes floating in space next to each other are not a Platonic solid.)

Theorem 23.1. *There are only five possibilities for the pair (p, q) in a Platonic solid.*

Proof. We can write down two equations for n (the number of vertices), m (the number of edges), and r (the number of faces) in terms of p and q.

- The graph is a q-regular graph, so by the handshake lemma (Lemma 4.1), $nq = 2m$.
- Every face has length p, so by the face length formula (Lemma 21.3), $rp = 2m$.

We also have Euler's formula (Theorem 21.4): $n - m + r = 2$. Replacing n by $\frac{2m}{q}$ and r by $\frac{2m}{p}$, we get

$$\frac{2m}{q} - m + \frac{2m}{p} = 2 \implies \frac{1}{q} - \frac{1}{2} + \frac{1}{p} = \frac{1}{m}.$$

From here, the constraint that lets us narrow down the pairs (p, q) is that $\frac{1}{m} > 0$. Therefore $\frac{1}{q} - \frac{1}{2} + \frac{1}{p} > 0$, or $\frac{1}{p} + \frac{1}{q} > \frac{1}{2}$.

How can we get a total bigger than $\frac{1}{2}$ here? Let's do casework on p:

- If $p = 3$ (every face is a triangle) then $\frac{1}{q} > \frac{1}{2} - \frac{1}{p} = \frac{1}{6}$, so $q < 6$.

  We can have $q = 3$ (three triangles meet at every vertex), giving us the tetrahedron.

  We can have $q = 4$ (four triangles meet at every vertex), giving us the octahedron.

  We can have $q = 5$ (five triangles meet at every vertex), giving us the icosahedron.
- If $p = 4$ (every face is a square) then $\frac{1}{q} > \frac{1}{2} - \frac{1}{p} = \frac{1}{4}$, so $q < 4$.

  We can have $q = 3$ (three squares meet at every vertex), giving us the cube.
- If $p = 5$ (every face is a pentagon) then $\frac{1}{q} > \frac{1}{2} - \frac{1}{p} = 0.3$, so $q < \frac{1}{0.3} = 3\frac{1}{3}$.

  We can have $q = 3$ (three pentagons meet at every vertex), giving us the dodecahedron.

These are the only possibilities: if $p \geq 6$, then not even $q = 3$ is small enough for the inequality $\frac{1}{p} + \frac{1}{q} > \frac{1}{2}$ to hold. □

In each of these cases, once we know p and q, we can solve for n, m, and r.

Question: What must n, m, and r be if $(p, q) = (3, 4)$?

Answer: Starting from Euler's formula $n - m + r = 2$, we can substitute $n = \frac{2m}{q} = \frac{1}{2}m$ and $r = \frac{2m}{p} = \frac{2}{3}m$ to get $\frac{1}{2}m - m + \frac{2}{3}m = 2$, or $\frac{1}{6}m = 2$. Therefore $m = 12$, and now we can back-substitute to get $n = \frac{1}{2}m = 6$ and $r = \frac{2}{3}m = 8$. This is the octahedron: it has $n = 6$ vertices, $r = 8$ faces, and $m = 12$ edges.

In general, the equation $\frac{1}{q} - \frac{1}{2} + \frac{1}{p} = \frac{1}{m}$, which can be rearranged to get $m = \frac{1}{1/p+1/q-1/2}$. Then $n = \frac{2m}{q}$ and $r = \frac{2m}{p}$ tells us the number of vertices and the number of faces. Here is the complete table (where $\langle q \rangle \times n$ stands for the sequence $q, q, \ldots, q$ of length n):

	Tetrahedron	Cube	Octahedron	Dodecahedron	Icosahedron
Number of vertices	4	8	6	20	12
Degree sequence	$\langle 3 \rangle \times 4$	$\langle 3 \rangle \times 8$	$\langle 4 \rangle \times 6$	$\langle 5 \rangle \times 12$	$\langle 3 \rangle \times 20$
Number of edges	6	12	12	30	30
Number of faces	4	6	8	12	20
Face types	$\triangle \times 4$	$\square \times 6$	$\triangle \times 8$	$\pentagon \times 12$	$\triangle \times 20$

I promised to mention a quibble I have with calling this a complete classification of the Platonic solids. Well, first of all, it is only a classification of the skeleton graphs of those solids: we have not (and will not) engage with the 3D geometry. Even so, one thing is missing.

Question: How could there, conceivably, be another regular planar graph where every face in a plane embedding has the same length, other than the ones in Figure 23.2?

Answer: It's possible that there are several different non-isomorphic graphs corresponding to a single pair (p, q).

So is there a second icosahedron where the faces attach differently? There is not, but that takes some effort to prove. In principle, it's a finite problem: up to isomorphism, there are only finitely many graphs with 20 or fewer vertices, and we could simply check them all and verify that none of them have the regularity properties we asked for.

It would be nice to have a more elegant argument, though. For the smaller Platonic solids, this is achievable. For example, the pair $(p, q) = (3, 3)$ must correspond to a 3-regular, 4-vertex graph, and there is only one such graph: the complete graph K_4. Slightly more complicated, but similar arguments work for $(p, q) = (3, 4)$ and $(p, q) = (4, 3)$; since I know it's possible to handle these two cases in a slick way, I will leave it for you to discover in the exercises at the end of this chapter. I am not aware of any arguments for $(p, q) = (3, 5)$ and $(p, q) = (5, 3)$ that do not require a substantial amount of suffering, so I will not make you deal with those cases.

23.3 Dual graphs

If you look at the table counting vertices, edges, and faces in the Platonic solids, you may notice an interesting pattern: the five solids can be divided into two and a half pairs for which these counts are related. The triple (n, m, r) counting the vertices, edges, and faces is $(8, 12, 6)$ for the cube and it is $(6, 12, 8)$ (the reverse) for the octahedron; it is $(20, 30, 12)$ for the dodecahedron and $(12, 30, 20)$ (the reverse) for the icosahedron.

Question:	Why "two and a half" pairs?
Answer:	The tetrahedron can be paired with itself, because it has the same number of vertices and faces.

There is another correspondence between vertices and faces that you may have noticed before. The face length formula for plane embeddings is very similar to the handshake lemma: if $F_1, F_2, \ldots, F_r$ are the faces and $x_1, x_2, \ldots, x_n$ are the vertices, then

$$\sum_{i=1}^{r} \text{len}(F_i) = 2m = \sum_{i=1}^{n} \deg(x_i).$$

Is this a coincidence?

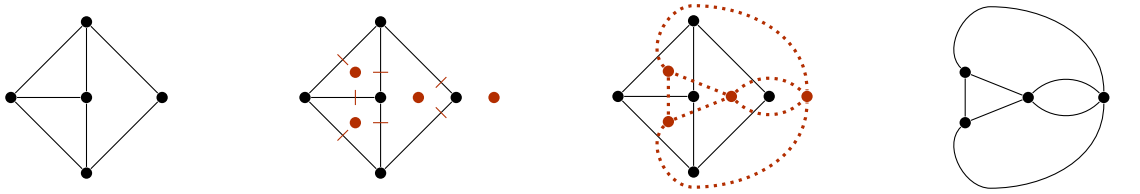
Mathematicians should always be on the lookout for such "coincidences", because it often turns out that they reveal a deeper idea. In this case, it leads to the definition of dual graphs.

The **dual graph** of a plane embedding is not really a graph, but a multigraph whose vertices are the faces of the plane embedding; for every edge in the plane embedding, there is an edge in the dual graph between the faces that meet there. For example, Figure 23.4d shows the dual graph of the plane embedding in Figure 23.4a.

Question:	The definition mentions that the dual graph is actually a multigraph. When does it have loops?
Answer:	When the original plane embedding has an edge with the same face on both sides. A vertex of degree 1 guarantees that this will happen, though it is not the only way.

Question:	What about parallel edges?
Answer:	The dual graph has parallel edges when two faces border each other along multiple edges. A vertex of degree 2 guarantees that this will happen, though it is not the only way.

The mere existence of the dual graph, carefully defined, is enough to derive the face length formula as a consequence of the handshake lemma. For relationships like those between the Platonic solids to holds, something more has to happen: the dual graph must itself be a planar graph (or multigraph).



(a) A plane embedding (b) Careful setup (c) Drawing the edges (d) The dual graph

Figure 23.4: The dual graph of a plane embedding

Proposition 23.2. *The dual graph of a plane embedding is also planar.*

Proof. Figure 23.4 shows the process of carefully constructing the dual graph in such a way that we get a plane embedding of the dual graph at the same time.

Begin by drawing a dual vertex somewhere in the interior of each face, and marking a crossing point somewhere in the middle of each edge. This is shown in Figure 23.4b.

Next, to draw a dual edge between the dual vertices inside faces F_i and F_j, we draw a curve from the dual vertex inside F_i, to the crossing point on the edge that F_i and F_j share, to the dual vertex inside F_j.

We still need to be careful to avoid crossings, but the setup means we need to be less careful. In order to know that this construction can always be carried out, we only need to know one thing: inside a face F_i, we can draw non-intersecting curves from the dual vertex to all the crossing points on the boundary of F_i. This is a "local" geometric claim that doesn't require us to consider the plane embedding as a whole. □

There are several properties that the dual graph isn't required to have, but that it will have in sufficiently nice cases. For example, strictly speaking, it is incorrect to refer to the "dual graph of G", where G is a planar graph. The dual graph is defined based on a plane embedding of G, not based on G itself. Sometimes we can get legitimately different dual graphs by choosing a different plane embedding, though the graph in Figure 23.4a and the skeleton graphs of the Platonic solids do not allow this.

Question:	From the plane embeddings we've seen so far, can you give an example where two different ones are guaranteed to give non-isomorphic dual graphs?
Answer:	In Chapter 21, Figure 21.5a and Figure 21.5b showed two plane embeddings of the same graph where the faces had different lengths. The dual graphs we get from these embeddings will not be isomorphic, because their vertices will have different degrees.

In Figure 23.4c, only the color indicates which edges are edges of the original plane embedding, and which edges are edges of the dual graph. Here, the dual relationship holds in both directions: each vertex of the original plane embedding lies in a face of (the plane embedding of) the dual graph. In such a scenario, taking the dual graph twice brings us back to where we started,

up to isomorphism. However, it is not guaranteed to happen; it is even possible for a plane embedding to have more vertices than the dual graph has faces.

Given a connected plane embedding with n vertices, m edges, and r faces, however, the dual graph will always have n faces (as well as m edges and r vertices). The number of edges and vertices in the dual graph follows from the definition, but the number of faces follows from Euler's formula. Applied to the original plane embedding, it tells us that $n - m + r = 2$. Meanwhile, if we suppose that a plane embedding of the dual graph has n' faces, then $r - m + n' = 2$; together, these two equations imply that $n = n'$.

Question:	Must the dual graph of a plane embedding always be connected?
Answer:	Yes: given any two faces F and F' (which are vertices of the dual graph), draw any curve from the inside of F to the inside of F', only making sure it does not pass through any vertex. The faces that the curve passes through form an $F - F'$ walk in the dual graph.

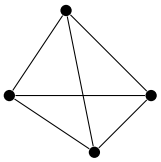
Finally, there is more to the story of duality in the case of Platonic solids (and some other polyhedra). For the skeleton graph of a polyhedron, we can construct a dual graph in a way that reflects the geometry of the polyhedron, by doing the following:

1. For the dual vertex corresponding to a face F of the polyhedron, draw a point in the geometric center of face F.
2. For the dual edge connecting adjacent faces F and F', draw a line segment between the two points in the centers of F and F'.

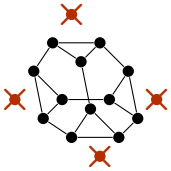
Because the dual vertices are adjacent exactly when the corresponding faces are adjacent, this is still a geometric realization of the dual graph. (It is not a plane embedding because it's all happening in three dimensions.)

This kind of dual turns the cube into the octahedron (and vice versa) geometrically, not just graph-theoretically; the same is true for the icosahedron and dodecahedron!

Question:	In general, though, this construction is not guaranteed to turn a polyhedron into another polyhedron. What could go wrong?
Answer:	The resulting points and line segments do not necessarily form faces that lie flat! If the dual graph has a face of length 4 or more, then the points on the boundary of that face might not end up lying on a single plane.



(a) Tetrahedron



(b) Truncated tetrahedron

Figure 23.5: Truncating a tetrahedron

23.4 Archimedean solids

The definition of a Platonic solid is the most restrictive generalization of a regular polygon to three dimensions. A slightly less restrictive, and still very interesting, definition is that of an *Archimedean solid*.

These are convex polyhedra whose faces are all regular polygons, and whose vertices are all symmetric to each other (that is, for any two vertices, there is some rotation or reflection of the polyhedron that can move one to the other). Notably missing from this definition is any kind of symmetry between faces: in an Archimedean solid, the faces do not all have to be the same!

The truncated tetrahedron is one small example of an Archimedean solid. Geometrically, it is obtained as follows: start with a tetrahedron, and cut off each vertex a third of the way along its edge, as shown in the picture below. The truncation process is shown in Figure 23.5.

The truncated tetrahedron has two types of faces: four hexagons (left over from the original faces of the tetrahedron) and four triangles (from where the cuts were made).

As with the Platonic solids, we can at the very least determine the global face, vertex, and edge counts from a local description of what is happening at every vertex. Let's see how, using the truncated tetrahedron as an example. (Imagine that we don't have the diagram in Figure 23.5b to use as a reference.)

Every vertex of the truncated tetrahedron is a third of the way along some edge of the tetrahedron. The two faces that meet there are two hexagons (from the two faces of the tetrahedron that met along that edge) and one triangle (from the cut that created that vertex). This is all the "local information" we will need. The variables we will solve for are:

1. m, the number of edges.
2. n, the number of vertices.
3. r_3, the number of 3-sided faces (triangles).
4. r_6, the number of 6-sided faces (hexagons).

We will need four equations, because there are four variables. Euler's formula is one of them: it tells us that $n - m + (r_3 + r_6) = 2$. It is tempting to use the face length formula, which tells us that $3r_3 + 6r_6 = 2m$, but this is less convenient because it involves three variables; instead, we take the handshake lemma, which tells us that $3n = 2m$. For the other two equations, we use the the local information about what happens at each vertex.

Question:	If there are n vertices, and each vertex the corner of one of the r_3 triangles, what is r_3 in terms of n?
Answer:	There is a 3-to-1 correspondence between vertices and triangles: each vertex has 1 triangle, but each triangle has 3 vertices. So $n = 3r_3$.

Question:	What about the relationship between n and r_6?
Answer:	Here, the correspondence is 6-to-2: each vertex has 2 hexagons that meet there, and each hexagon has 6 vertices at its corners. So $2n = 6r_6$.

In general, such equations are determined by two quantities: the number of sides each type of face has, and the number of faces of that type that meet at each vertex.

Now we can write Euler's formula solely in terms of n, by replacing each variable by a multiple of n: since $m = \frac{3}{2}n$ and $r_3 = r_6 = \frac{1}{3}n$, Euler's formula turns into

$$n - \frac{3}{2}n + \frac{1}{3}n + \frac{1}{3}n = 2.$$

When we simplify, we get $n = 12$. Therefore $m = \frac{3}{2}n = 18$, $r_3 = \frac{1}{3}n = 4$, and $r_6 = \frac{1}{3}n = 4$: exactly the parameters of a truncated tetrahedron!

There is a way to make this process more systematic, while also generalizing it to be able to deal with even less regular polyhedra.

To do so, we define the *angle defect* at a corner of a polyhedron to be 2π minus the sum of the angles of the polygons meeting at that corner. (We'll work in radians from now on; in degrees, we'd take $360°$ instead of 2π.) Since the angle defect would always be 0 if the corner were flat, this is a measure of how much the polyhedron "bends" at a corner.

For a convex polyhedron (or any polyhedron for which Euler's formula holds), no matter how many vertices there are, the total amount of "bend" must be the same. This was originally shown by René Descartes, the inventor of coordinate geometry [33]:

Theorem 23.3. *In any convex polyhedron, the sum of all angle defects is* 4π.

Proof. This could be done by solving a system of equations, but there is a more elegant proof by a strategy called the "discharging method". We take the skeleton graph of the polyhedron, and put a "charge" of $+2\pi$ on each vertex, $+2\pi$ on each face, and -2π on each edge. The total charge on the graph is $2\pi n - 2\pi m + 2\pi r$, and we've chosen our initial charges so that the total charge would simplify to 4π by Euler's formula.

In physics, positive and negative electric charges cancel. Probably. I'm not a physicist. In this proof, we will move around the charges we've placed on the polyhedron to cancel them, while not changing the overall sum. First, from each face, we move $+\pi$ charge on to each of its edges. This leaves each edge at charge 0; it started at -2π, but gained $+\pi$ from each of the two faces it borders. However, each face has now gone into the negatives: a face of length l now has charge $-(l-2)\pi$.

In an l-sided polygon, the sum of angles is $(l-2)\pi$ (as observed at the beginning of this chapter for regular polygons). So we can bring each face up to zero charge with a second transformation: for every corner of every face, if that corner makes an angle of θ on that face in the polyhedron, we move θ charge from the vertex at that corner to the face.

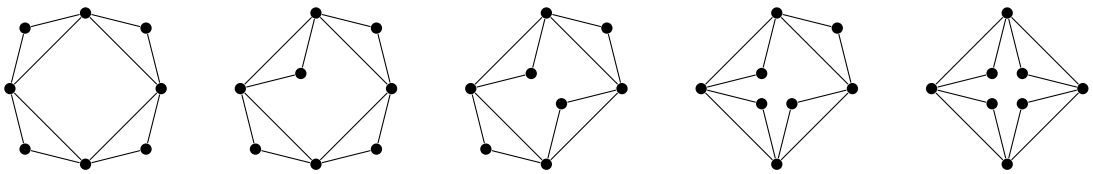
When we're done, the faces and edges all have charge 0, while the remaining charge at each vertex is exactly the angle defect. However, the sum of the charges has remained at 4π throughout, proving the formula. □

Theorem 23.3 can be used to quickly count the vertices in any Archimedean solid. For example, in the truncated tetrahedron, a regular triangle and two regular hexagons meet at each vertex, forming one angle of measure $\frac{\pi}{3}$ and two angles of measure $\frac{2\pi}{3}$. This means that the angle defect at each vertex is $2\pi - \frac{\pi}{3} - 2(\frac{2\pi}{3}) = \frac{\pi}{3}$. The total angle defect is 4π, so there must be $\frac{4\pi}{\pi/3} = 12$ vertices.

23.5 Practice problems

1. Draw a plane embedding of the skeleton graph of the dodecahedron.
2. The skeleton graph of the icosahedron is pancyclic: it has a cycle of every length from 3 to 12. Verify this by finding a cycle of each length.
3. Determine the dual graph of any plane embedding of any n-vertex tree, up to isomorphism.

   (This example goes to show that two planar graphs with isomorphic duals are not, themselves, necessarily isomorphic.)
4. Here are five different plane embeddings of a graph G:

   For each plane embedding, draw the dual graph. Determine which of these graphs are isomorphic to each other, and which are not.
5. By going from a plane embedding to a spherical embedding and back to a plane embedding (using the projection technique in Figure 23.3), prove that for any face F of a plane embedding of G, there is a plane embedding of G where F is the outer face.
6. A uniform n-gonal prism is a prism with $n+2$ faces: two regular n-gons on the top and bottom, and n squares around the sides.

   a) Draw a plane embedding of the skeleton graph of a uniform n-gonal prism where n is some very big number—like 6. Explain how to draw such a plane embedding for any value of n.

   b) Draw the dual graph of the plane embedding you drew in part (a). Describe the structure of this dual graph for arbitrary values of n.

c) Geometrically, what does the dual polyhedron of the n-gonal prism look like?

7. a) (AIME 2004) A convex polyhedron P has 26 vertices, 60 edges, 36 faces, 24 of which are triangular, and 12 of which are quadrilaterals. A space diagonal is a line segment connecting two non-adjacent vertices that do not belong to the same face. How many space diagonals does P have?

   b) How many of the numbers given in part (a) are redundant information?

8. Here are few more questions about Archimedean solids.

   a) An icosidodecahedron is an Archimedean solid with 12 pentagonal faces (like a dodecahedron) and 20 triangular faces (like an icosahedron). How many vertices and edges does it have? How many faces of each type meet at each vertex?

   b) A snub cube is an Archimedean solid with four triangles and one square meeting at every vertex. How many vertices and edges does it have, and how many faces of each type?

   c) What about the truncated icosidodecahedron, in which a 4-sided face, a 6-sided face, and a 10-sided face meet at every vertex?

9. Suppose that G is an n-vertex planar multigraph such that (for at least one plane embedding of G) the dual graph is isomorphic to G. (We call such a multigraph self-dual.)

   a) How many edges must G have, in terms of n?

   b) Find an example of such a graph G for all $n \geq 2$.

10. In this problem you will prove that the standard octahedron and cube are the only possible Platonic solids with their parameters, at least from the point of view of graph theory.

    a) It follows from Theorem 23.1 that any Platonic solid with $(p, q) = (3, 4)$ is a 4-regular 6-vertex graph. Prove that there is only one such graph (up to isomorphism).

    b) It follows from Theorem 23.1 that any Platonic solid with $(p, q) = (4, 3)$ is a 3-regular 8-vertex graph. Unfortunately, there are multiple such graphs. However, the graph must also have a plane embedding in which every face has length 4, and by a practice problem at the end of Chapter 21, it must be bipartite.

    Prove that there is only one bipartite 3-regular 8-vertex graph (up to isomorphism).

24 Coloring maps

The purpose of this chapter

I have to write about map coloring at some point in this textbook; there's no way around it. The map coloring problem is not only the reason graph coloring was invented; it's also a big part of the story behind the invention of graph theory as a coherent discipline. That's also why there's more history in this chapter of the textbook than in most other chapters.

I don't usually spend a whole lecture on map coloring when teaching a course on graph theory, because there's not enough time. I think going up to just Theorem 24.4 is a reasonable minimum: it is a good way to see how the properties of planar graphs we already know can be applied, and it's a nice application of coloring graphs greedily. The proof of Theorem 24.5 I've included is a bit shorter than the usual one, specifically in case you've decided to spend a bit more time on the topic of map coloring, but not too much time. (As a side note, the edge contraction in the proof has a particularly nice representation if we are coloring a map, not a graph: then, we just erase the borders region x has with y_i and y_j.)

The last two sections cover two special topics that are less frequently seen in graph theory courses, but which I think are very interesting. (I think it's fascinating how—for the second time in our study of planar graphs!—finding a Hamilton cycle can help us solve a seemingly unrelated problem.) Heawood's empire coloring problem is a good example of how we can arrive at new mathematical questions by examining the assumptions in our simplified models.

In addition to the previous chapters in this book, this chapter heavily relies on Chapter 19 (naturally). On the other hand, the section on the use of Hamilton cycles in coloring maps will not ask you to know much more from Chapter 17 than the definition of a Hamilton cycle.

24.1 Coloring maps

In Chapter 1, we visited Switzerland; let us return there.

Figure 24.1 shows a map of the cantons of Switzerland. To make the borders between cantons easy to see, we give them different colors. We are willing to reuse colors, but we would like two cantons to have different colors when they share a border. (It would be fine for two cantons to have the same color if they only share a corner; this situation is rarely found in maps.) How many different colors do we need to color Switzerland—and is there a number of colors that would be enough to color any map, no matter how complicated?

Let the **graph of adjacencies** of a map be the graph whose vertices are the regions we must color, with an edge between vertices that share a border. Then what we are looking for is a (proper) coloring of the graph of adjacencies: we've arrived back at the the graph coloring

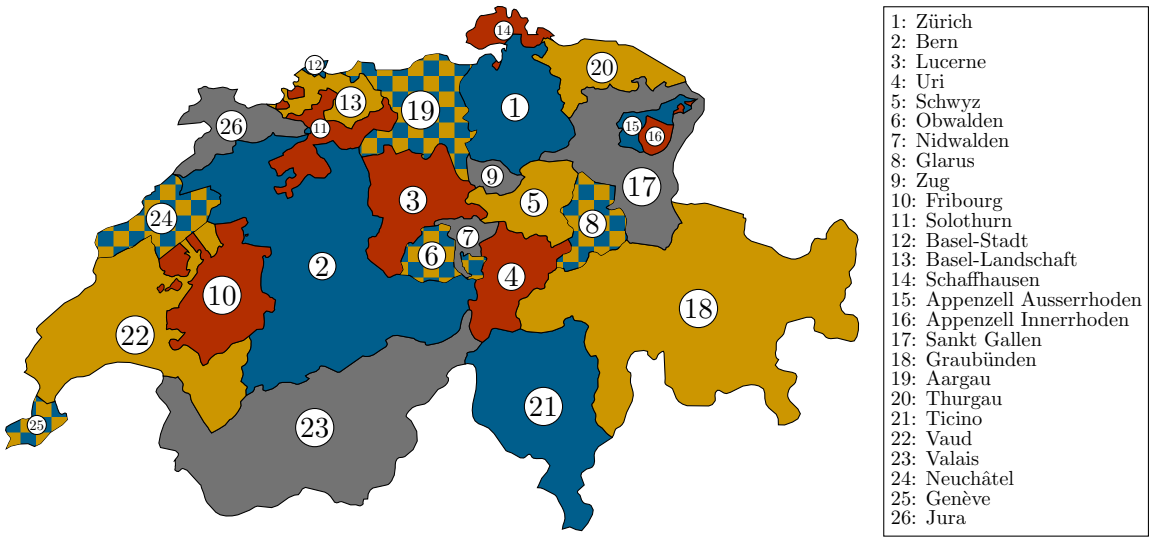


Figure 24.1: A map of Switzerland (coordinates from [92])

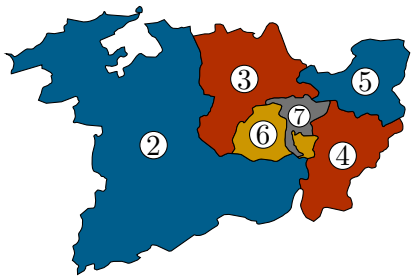
problem studied in Chapter 19! Historically, in the middle of the 19$^{\text{th}}$ century, this was the first graph coloring problem studied. Graph theory had yet to be codified as a discipline at the time, though some problems were studied individually which are now the domain of graph theory. The book *Four Colors Suffice* by Robin J. Wilson [107] gives a history of the problem, and is also the source for most of my historical claims in this chapter.

Why are we only looking at the map coloring problem now? The reason is that for maps with reasonable, well-behaved regions, the graph of adjacencies is planar. We can show this by the same argument we used to prove Proposition 23.2 that the dual graph of a plane embedding is planar. In fact, we can think of the graph of adjacencies as the dual of a plane embedding whose edges are exactly the borders drawn in the map. (We add vertices to give these edges suitable endpoints; aside from a few edge cases, vertices are mostly only necessary where three borders meet.)

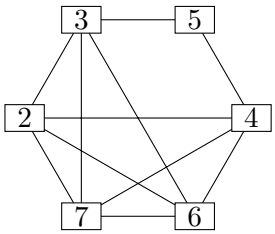
I said "maps with reasonable, well-behaved regions" because the real world is messy, and not all actual maps translate to planar graphs as neatly. In fact, Switzerland already provides an example of this: its cantons are not what I would call reasonable and well-behaved!

Question:	What properties do the Swiss cantons have that the faces of a plane embedding shouldn't?
Answer:	They're often not connected regions! Four cantons have multiple pieces separated by another canton: Obwalden (6), Fribourg (10), Solothurn (11), and Vaud (22).

In fact, we can prove that the graph of adjacencies between Swiss cantons is not a planar graph. This follows from Kuratowski's theorem (Theorem 22.7). It's enough to look at the cantons shown in Figure 24.2a: because canton 6 (Obwalden) has two pieces, there are a few more borders between these than would be possible in a reasonable map. The graph of adjacencies between these 6 cantons contains a subdivision of the complete graph K_5, which is shown in



(a) A map of a few Swiss cantons



(b) A subdivision of K_5 in Switzerland

Figure 24.2: Why Switzerland is not planar

Figure 24.2b: 9 of the 10 edges between cantons $\{2, 3, 4, 6, 7\}$ are present directly, and instead of the edge $\{3, 4\}$ there are edges $\{3, 5\}$ and $\{5, 4\}$ through 5, an extra vertex. (Edge $\{5, 7\}$ is also present in the graph, but is not shown in Figure 24.2b because it is not part of the subdivision.) Since the subgraph in Figure 24.2b is not planar, the entire graph of adjacencies between Swiss cantons cannot be planar, either.

This is an extremely practical example of a misbehaving map; it is also possible to give another example that is extremely theoretical. Hud Hudson showed [58] that if we allow fractal regions with a finite area and infinite perimeter (considering them to share a border if both regions get arbitrarily close to that border), then even connected regions can touch in extremely non-planar ways. There is no limit to the number of colors that might be necessary to color such a fractal map.

From now on, we will assume that our maps are reasonable and well-behaved: that the borders between regions obey the same restrictions as edges in a planar graph (there are no fractal borders) and that the objects we are coloring are nothing more than the connected regions separated by those borders (there are no regions with multiple pieces). With these caveats, the problem of coloring maps is exactly equivalent to the problem of coloring planar graphs.

24.2 The four color theorem

What could stop us from coloring a map with k colors, for some integer k? The most straightforward kind of obstacle is a set of k regions among which any two are adjacent: a k-vertex clique in the graph we are coloring. It's possible to draw a map with four regions that all touch; this is the simplest possible example of a map which requires four colors.

Question:	Is it possible to draw a map with 5 regions that all touch?
Answer:	No, because the graph of adjacencies would be K_5, which is not a planar graph.

Question:	Does this prove that four colors are enough to color any map?
Answer:	No! It is possible to have a graph with no 5-vertex clique which is still not 4-colorable.

If you've been reading this book carefully and in order, Chapter 19 should already have prepared you for the idea that the chromatic number $\chi(G)$ and the clique number $\omega(G)$ are very different. Forgetting this is one of the most common mistakes people make when they first think about coloring planar graphs.

From a pedagogical point of view, I can't help but think that it's a shame that the following theorem really is true:

Theorem 24.1 (Four color theorem)**.** *If G is a planar graph, then $\chi(G) \leq 4$.*

The road to proving this theorem has been a long one. The mathematical question of whether four colors are enough to color any map was first asked by Francis Guthrie in 1852. Later in the 19$^{\text{th}}$ century, it received a great deal of mathematical attention, which came with multiple incorrect solutions to the problem.

Some incorrect solutions were the sort of careless error that confuses the chromatic number with the clique number. I assume that even at the time, there were also many attempts to disprove the theorem, from people who drew a very complicated map and could not find a way to color it with four colors; there are still such attempts. (At the same time, there was very persuasive experimental evidence for the four color theorem: every real-world and imaginary map that was tried could in fact be colored with four colors.)

There were also more notable attempts. Later on in this chapter, I will mention two attempts at proving the four color theorem that were eventually found to be false, but contributed a lot to graph theory, even so. Alfred Kempe's approach had a subtle error, but it was still enough to give a proof of the weaker bound in Theorem 24.5, and the ideas in Kempe's proof have since been used to solve other problems about graph coloring. And I suspect that Peter Tait's use of Hamilton cycles to try to color maps was one of the big reasons why 19$^{\text{th}}$ century mathematicians continued to study Hamilton cycles, before practical uses of them were found!

Eventually, a proof of the four color theorem was found, but even that was controversial. Kenneth Appel and Wolfgang Haken, in a long effort from 1972 to 1976, found a proof that relied on using a computer to verify 1936 different configurations [4, 5]. How can any number of finite cases be used to prove a theorem about the infinite variety of possible maps? The idea, simplified, is that any possible map would be guaranteed to contain one of the "reducible configurations" that Appel and Haken found. The configurations were not just substructures that could be colored: they could be replaced by smaller ones without hurting colorability, which allows for a proof by induction on the number of regions in the map.

A proof so long that it needed a computer to verify was controversial. Wouldn't it be invalidated by a single bug in the program? This was the fear at first, but by now, it's been nearly 50 years, and we can be a bit more confident, for a few reasons:

- A 1997 proof by Robertson, Sanders, Seymour, and Thomas, while still a computer-based proof, used fewer configurations and a simpler approach to reductions [90].
- In 2005, Georges Gonthier formalized the entire proof in the Coq proof assistant [38]. Though the proof is still checked by computer in this case, it does not rely on rules specific to map coloring, only on formal logic, so we can be more confident in the computer.
- It's been nearly 50 years! If there were something incurably wrong, surely we'd have discovered it by now.

Even if we believe the computer, there is still a reason why we might want a short, human-readable proof (which still has not been found). Such a proof would certainly contain new ideas that we can apply to solve other, more difficult problems! This is not to say that the computer-based proofs have no such insights; the whole approach of reducible configurations, refined several times, is mathematically interesting. But checking a large number of cases by computer does not tell us whether there is some underlying simple reason why all those cases would work.

You might have guessed by now that I will not show you a proof of the four color theorem in this chapter. We will, however, look at the arguments involved in proving two weaker bounds.

24.3 Greedy coloring

The greedy coloring algorithm goes through the vertices and gives each one a color not already used on its neighbors. This algorithm is guaranteed to find a coloring of the graph, but the number of colors used depends on the order in which we color the vertices. The best thing we can say about the greedy algorithm in general is that if a graph G has maximum degree $\Delta(G)$, it will never use more than $\Delta(G) + 1$ colors.

Question:	Does give us any kind of universal upper bound for planar graphs?
Answer:	No: the maximum degree of a planar graph can be arbitrarily high. For example, the star graph S_n is a tree with $n-1$ leaf vertices adjacent to one central vertex; it is planar (like all trees) but has maximum degree $n - 1$.

For planar graphs, it is possible to use some of the theory we've already developed to order the vertices more intelligently. The beginning is a bound on the minimum degree of a planar graph: though planar graphs can have vertices of very large degree, this cannot be true of every vertex.

Lemma 24.2. *Every planar graph G has minimum degree $\delta(G) \leq 5$.*

Proof. This is true for every planar graph with at most 6 vertices because at that point, you can't have any degrees bigger than 5.

For planar graphs with $n \geq 3$ vertices and m edges, Theorem 22.2 tells us that $m \leq 3n - 6$. However, if every vertex had degree 6 or more, then we would have $m \geq \frac{1}{2}(6n) = 3n$ by the handshake lemma (Lemma 4.1), and we cannot have $3n \leq 3n - 6$. Therefore not all vertices have degree 6 or more: there must be a vertex with degree 5 or less. □

An alternate way to phrase the proof would be to look at the average degree of a vertex, rather than the total number of edges.

Question:	What is the average degree of a graph with n vertices and $m \leq 3n-6$ edges, and how does this help us?
Answer:	It is given by $\frac{2m}{n} \leq \frac{2(3n-6)}{n}$, which simplifies to $6-\frac{12}{n}$. Since $6-\frac{12}{n} < 6$, the average degree is always less than 6. Not all vertices can be above average, so there must be a vertex of degree less than 6.

It is convenient to have a vertex of small degree, because we can leave it to be colored last: even if the worst should happen and all its neighbors have different colors, we will still be able to pick a color for it. For example, if we have 6 colors available, and we leave a vertex of degree 5 until the end, it will always be possible to give it a color.

Question:	If $\delta(G) \leq 5$, is that enough to know that $\chi(G) \leq 6$: that G is 6-colorable?
Answer:	No! For example, we could start with K_{100} and add a new vertex of degree 5 (or degree 0). That low-degree vertex can be left until the end, but we'll still need 100 colors to color K_{100}.

In the case of a planar graph, however, we know more. If we remove a vertex of minimum degree, what we're left with is a smaller planar graph, and Lemma 24.2 also applies to that smaller planar graph. That graph, too, has a vertex of degree 5 or less, which is enough to give us a proof by induction.

Lemma 24.3. *Every n-vertex planar graph G has a vertex ordering $x_1, x_2, \ldots, x_n$ in which each vertex is adjacent to at most 5 of the vertices that come before it.*

Proof. We will prove this by induction on n. When $n \leq 6$, any vertex ordering will do.

Assume that the lemma is true for all $(n-1)$-vertex planar graphs, and let G be an n-vertex planar graph. By Lemma 24.2, $\delta(G) \leq 5$; let x be a vertex with $\deg_G(x) \leq 5$.

Apply the induction hypothesis to find a vertex ordering $x_1, x_2, \ldots, x_{n-1}$ of $G - x$. We can extend it to a vertex ordering of G by setting $x_n = x$. For all $i \leq n-1$, x_i has fewer than 5 neighbors among $\{x_1, x_2, \ldots, x_{i-1}\}$ by the induction hypothesis. Meanwhile, x_n has fewer than 5 neighbors among $\{x_1, x_2, \ldots, x_{n-1}\}$ because it has fewer than 5 neighbors total.

By induction, we can find such a vertex ordering for all n. □

Using this lemma, we can prove our first bound on the chromatic number of arbitrary planar graphs!

Theorem 24.4. *If G is a planar graph, then $\chi(G) \leq 6$.*

Proof. Let $x_1, x_2, \ldots, x_n$ be the vertex ordering given by Lemma 24.3, and let C be a set of 6 colors. For $i = 1, 2, \ldots, n$ in that order, give x_i an arbitrary color in C not already used on its neighbors.

Why is this possible? Because x_i only has 5 neighbors in the set $\{x_1, x_2, \ldots, x_{i-1}\}$, and these are the only vertices already given a color when we get to x_i. So at most 5 of the 6 colors in C have been used on the neighbors of x_i, and there is still a color left to choose.

When we're done, the coloring we get is proper, because we never give a vertex a color used on an adjacent vertex. For every edge $x_i x_j$, where $i < j$, we already knew the color of x_i when we got to x_j, and we made sure that x_j would be given a different color. □

24.4 Five colors

Suppose we want to go one step further, and prove that $\chi(G) \leq 5$ for all planar graphs G.

Question:	What would go wrong if we tried the methods in the previous section to prove this?
Answer:	We would encounter planar graphs with minimum degree 5. Here, we cannot leave any vertex for last and forget about it; its neighbors might end up with 5 different colors, and we would have no color left to use on the last vertex.

Question:	Are there, in fact, any planar graphs with minimum degree 5?
Answer:	Yes, and we've seen them already: the skeleton graph of the icosahedron is one example.

We can still try to find an recursive algorithm for 5-coloring planar graphs: given a planar graph G and a vertex x of minimum degree, we 5-color $G - x$ and then complete the result to a coloring of G. However, if $\deg_G(x) = 5$, we will need to do one of two things:

1. Before we color $G - x$, modify it somehow to ensure that we'll be able to put back x and give it a color.
2. After we color $G - x$, modify the coloring somehow to ensure that we'll be able to put back x and give it a color.

Historically, the first proofs of Theorem 24.5 used the second approach, modifying the coloring using subgraphs called Kempe chains; this idea is similar to the recoloring strategy we used in the proof of Vizing's theorem (Theorem 20.6). Kempe chains are named after Alfred Kempe, who used the idea in 1879 in a proposed proof of the four color theorem. In 1890, Percy Heawood found a subtle error in Kempe's proof, but the argument via Kempe chains was still the first argument showing that five colors are always enough to color any map.

We will instead take the first approach, which is a shorter and more direct argument. This proof is due to Paul Kainen [60] and is a much later invention.

Theorem 24.5. *If G is a planar graph, then $\chi(G) \leq 5$.*

Proof. We induct on n, the number of vertices of G. (This will be a strong induction, because sometimes we'll need to go back from n to $n-2$, not just $n-1$.) If $n \leq 5$, then of course G has a coloring with at most 5 colors; this gives us our base cases.

Now assume that all planar graphs with fewer than n vertices have colorings with at most 5 colors, and let G be an n-vertex planar graph. By Lemma 24.2, G has a vertex x of degree at most 5.

If in fact $\deg(x) \leq 4$, then we have lucked out. By induction, $G - x$ has a coloring with at most 5 colors. In that coloring, the neighbors of G use at most 4 of the colors, because there are at most 4 of them, so we can give x a color none of them use. The result is a coloring of G.

If $\deg(x) = 5$, let y_1, y_2, y_3, y_4, y_5 be the neighbors of x. The first thing we observe is that it's not possible for all of the edges between $y_1, \ldots, y_5$ to be present in G.

Question:	Why not?
Answer:	Then they'd form a copy of K_5 (a copy of K_6, in fact, when taken together with x), which we know is not planar; but G is a planar graph.

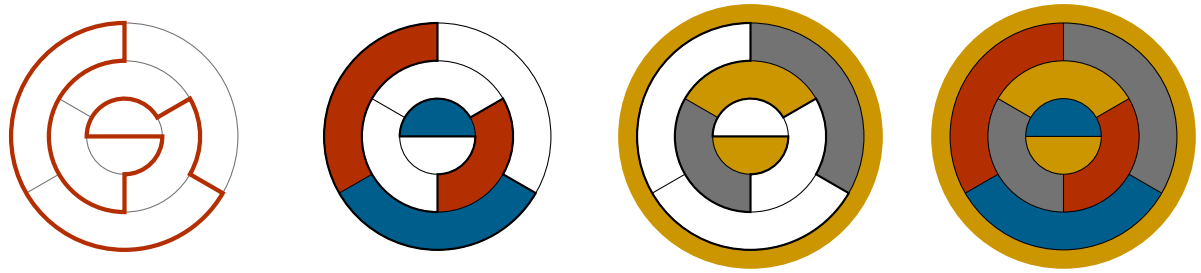
So let y_i and y_j be two of x's neighbors that are not adjacent in G. (We'll see why this matters soon!) Construct a graph H in the following steps:

1. Delete all edges from x except edges xy_i and xy_j.
2. Contract edges xy_i and xy_j, obtaining a new vertex z.

Deleting and contracting edges preserves planarity, so H is a planar graph on $n-2$ vertices. By our induction hypothesis, H has a 5-coloring. Extend this 5-coloring to $G - x$ by giving y_i and y_j the color of z.

Question:	Why is this a proper coloring?
Answer:	Every neighbor of y_i or y_j in $G - x$ was a neighbor of z in H, so it had a different color from the color used on z. Therefore in $G - x$, it has a different color from the color used on y_i and y_j. This is all that needs to be checked; for all other edges, both endpoints have the same color in $G - x$ and in H.

Question:	What would have gone wrong if y_i and y_j were adjacent?
Answer:	Then both endpoints of the edge y_iy_j would have the same color, so the coloring of $G - x$ would not be proper.



(a) A Hamilton cycle (b) Color the inside (c) Color the outside (d) The final result

Figure 24.3: Using a Hamilton cycle in the borders of a map to 4-color it

Now we have a 5-coloring of $G - x$ in which only 4 different colors are used on the neighbors of x, because two of them (y_i and y_j) have the same color. Once again, this lets us give x a color not used on any of its neighbors and obtain a coloring of G.

This proves the induction step, and so the theorem is true for planar graphs with any number of vertices. □

24.5 Hamilton cycles

In 1880, Peter Tait proposed several solutions of his own to the map coloring problem. At the time, Kempe's 1879 proof was generally accepted; Tait merely felt that the proof was too complicated, and wanted a simpler one. None of Tait's solutions worked out in the sense of giving an alternative proof, but several of them were successful in giving alternate avenues of attack on the problem.

Earlier, I mentioned that the graph of adjacencies of a map is the dual of a plane embedding whose edges are exactly the borders drawn in the map. However, we did not really consider this plane embedding as an object of study in its own right. Tait was the first to do so, and was able to get remarkably close to a proof of the four color theorem in this way.

Tait began by finding a Hamilton cycle in this plane embedding; for an example, consider the Hamilton cycle in Figure 24.3a. The Hamilton cycle divides the plane embedding into two parts, an inside and an outside. Tait's strategy was to color those two parts separately (see Figure 24.3b and Figure 24.3c) then combine the colorings, as shown in Figure 24.3d.

Why would this help? Well, Tait noticed that the inside and the outside of the cycle can each be colored with just two colors: four total! In fact, the graph of adjacencies on either side of the Hamilton cycle is a tree, and as we know (Proposition 13.3), all trees are bipartite, or in other words 2-colorable.

To see this, let H (for "half") be the graph of adjacencies on just one side of the Hamilton cycle. Each edge in H exists due to a border in the plane embedding where we drew the Hamilton cycle; both endpoints of that border lie on the cycle, because it's a Hamilton cycle. If a border is drawn between two points on a closed loop, it cuts that loop in half, separating the two parts of the side it's drawn on. Back in H, that makes the edge we were looking at a bridge. A connected graph in which every edge is a bridge is a tree.

Question:	Why is this not a proof of the four color theorem?
Answer:	The missing detail is that we don't yet have a reason to believe that the Hamilton cycle we are relying on exists!

In fact, it's pretty easy to draw a map in which we can't draw a Hamilton cycle along the borders. (Switzerland is an example: here, the borders aren't even connected!) Tait was aware of this, but still had hope to do it in all the worst cases, which would be enough to prove the four color theorem.

What are the worst cases? The phrase "worst case" is one you should generally watch out for in your proofs: if you're not careful, it can lead to unjustified assumptions. To properly use it, we should first explain how every other case is simpler than some "worst" case.

That's exactly what we can do here. Suppose we are trying to color an arbitrary planar graph G. If there is any edge xy such that adding xy to G produces another planar graph $G+xy$, then we might as well add it! Every coloring of $G+xy$ is also a coloring of G, with the extra restriction that x and y cannot be given the same color. And if we think we have a rule for coloring all planar graphs, that rule should be able to handle $G+xy$ just as well as G.

This means that it's enough to consider only maximal planar graphs, which cannot gain any new edge and still be planar. By Proposition 22.4, these are exactly the graphs whose plane embeddings are always triangulations.

Question:	What do we know about the dual of a triangulation?
Answer:	The condition that each face has length 3 turns into a condition that each vertex has degree 3. They are 3-regular graphs! (In fact, not all 3-regular planar graphs are the duals of triangulations; the graph must also be 3-*connected*, but we won't learn what that means until Chapter 26.)

Tait's conjecture was that the dual graph of every triangulation is Hamiltonian. If this were true, it would imply the four color theorem, by the strategy we pursued in Figure 24.3a. This seemed to be a viable strategy for a long time; it was not until 1946 that Tutte found a counterexample [99]. That counterexample is shown in Figure 24.4 and is known as the *Tutte graph*. (Here, the planar graph which cannot be colored using Tait's strategy is the dual of the Tutte graph; that is, we wish to color the faces in Figure 24.4. By the four color theorem, of course, this is still possible—just not via finding a Hamilton cycle.)

24.6 Coloring empires

In 1890, along with finding the flaw in Kempe's attempt at the four color theorem, Heawood asked [54]: what about maps like the map of Switzerland? That is, what about maps where a single country can have multiple disconnected parts, which must be given the same color?

If we don't put some kind of restriction on this ability, then there is no limit to the number of colors we might need.

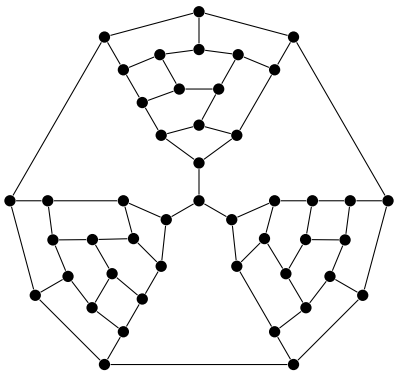


Figure 24.4: The Tutte graph

Question:	Why not?
Answer:	For example, give each country a tiny exclave in the middle of each other country's territory. Now this prevents those two countries from sharing a color, so every country will need a different color.

Heawood's approach was to express the problem in terms of the maximum number of parts a single country can have. Unfortunately, he did not come up with a clever name for such divided countries. This omission was corrected by Martin Gardner when writing about Heawood's work much later [36]; Gardner proposed the term *M-pire* for an empire whose territory consists of at most M disconnected pieces.

Let's begin by looking at the case $M = 2$. (This more or less describes Switzerland; though some Swiss cantons have more than 2 pieces, this does not contribute any additional adjacencies.)

Proposition 24.6. *A map of 2-pires can always be colored using at most 12 colors so that two 2-pires which share a border (in at least one of their territories) receive different colors.*

Proof. We can model a map of 2-pires in two ways:

1. With a graph (call it H) in which vertices are connected regions, and an edge exists whenever two regions share a border. Let there be n regions; then (as we've already seen) this graph is planar and has at most $3n - 6$ edges.
2. With a graph (call it G) in which vertices are the empires, and an edge exists between two vertices if some territory belonging to one empire borders some territory belonging to the other empire. This is the graph we actually want to color.

In G, there are also at most $3n - 6$ edges! That's because every edge in G must come from an edge in H (a border shared is a border shared in either representation) but every edge in H corresponds to at most one edge in G (a shared border only belongs to two empires).

Question:	Is it possible for G to have fewer edges than H?
Answer:	Yes: whenever one empire touches two of another empire's regions, that is represented by two edges in H, but only one edge in G.

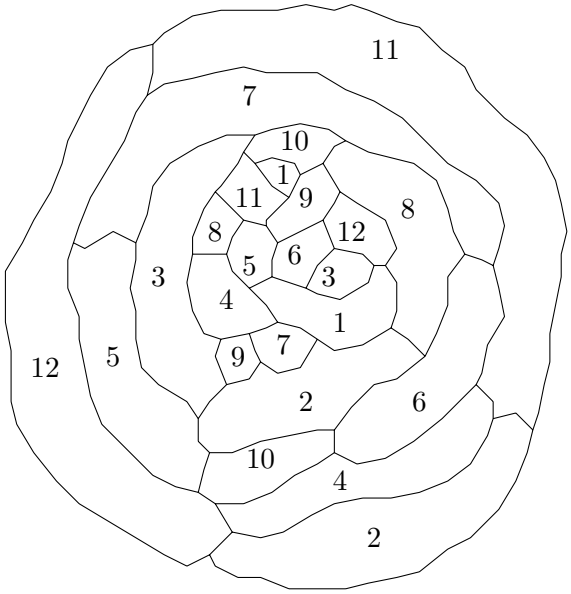


Figure 24.5: Heawood's map of 12 pairwise adjacent empires

Question:	What can we say about the number of vertices in G?
Answer:	It could be as high as n (if every 2-pire is actually an ordinary country), but cannot go below $n/2$: with at most 2 regions per empire, it takes at least $n/2$ empires to reach n regions.

The average degree in G is $\frac{2|E(G)|}{|V(G)|}$ by the handshake lemma, or at most $\frac{3n-6}{n/2} = 12 - \frac{24}{n}$. In particular, it is less than 12, so $\delta(G) < 12$.

What's more, if we delete a vertex x from G, then the remaining graph $G - x$ is also the graph of adjacencies of some map of 2-pires: just erase empire x from the map! This means that $\delta(G - x) < 12$, by the same argument, and in general, every subgraph of G will have a vertex of degree less than 12.

We can now color G by the same greedy argument as we used to prove Theorem 24.4, using 12 colors instead of 6. Leaving a low-degree vertex until the end guarantees that one of 12 colors will be available for it, since its neighbors have at most 11 different colors. □

Theorem 24.4 was considerably more pessimistic than the truth: it says that $\chi(G) \leq 6$ for all planar graphs G, but actually, it is also true that $\chi(G) \leq 4$. Does Proposition 24.6 have a similar flaw?

No: it turns out that it is the best possible bound! Figure 24.5 shows a map drawn by Heawood. Here, there are 12 empires (labeled 1 through 12), and any two of them share a border! For this map, it is impossible to use fewer than 12 colors, because all empires must have different colors; therefore there cannot be a general result better than Proposition 24.6.

Heawood gave an argument for the general case of M-pires, as well.

Question:	What upper bound does the same argument give for a map of M-pires?
Answer:	An upper bound of $6M$ colors: with n regions, there are still at most $3n - 6$ edges (less than $3n$) but the number of M-pires is at most n/M, so the average degree is less than $\frac{2(3n)}{n/M} = 6M$.

However, he was unable to find a map to show that this was the best upper bound when $M \geq 3$, and left this as a conjecture. This conjecture was resolved almost a century later, when Brad Jackson and Gerhard Ringel showed how to construct such maps for all $M \geq 2$ [59].

24.7 Practice problems

1. a) Find a 4-coloring of the faces of an icosahedron.

   b) Prove that 3 faces are not enough to color the faces of an icosahedron.

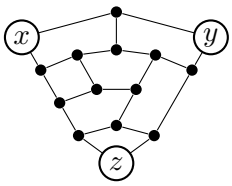
2. Even though Switzerland's graph of adjacencies is not planar, it is still 4-colorable! Find such a 4-coloring. (You may prefer to refer to the diagram in Figure 1.2 all the way back in the first chapter instead of a map of Switzerland.)

3. The mathematician August Ferdinand Möbius is perhaps best known for his mathematical study of the Möbius strip: the surface formed by taking a strip of paper and joining the two ends with a half-twist, so that the two sides of the paper are turned into one side. In regards to coloring maps, his contribution was to show that 5 connected regions on a map cannot all share borders with each other; an early form of proving that K_5 is not planar.

   Ironically, when drawing a map on a Möbius strip rather than in the plane, it is possible to have 5 regions all border each other. How can this be done?

   As a follow-up, see if you can do it for 6 regions.

4. The Tutte graph shown in Figure 24.4 is made up of three subgraphs isomorphic to the one shown below, plus a central vertex. The three vertices labeled x, y, and z in the diagram are used to connect the three subgraphs together: each x-vertex is joined to a y-vertex in a different subgraph to connect the three cyclically, and the three z-vertices are all joined to the center vertex of the Tutte graph.



   a) Show (by casework) that the graph above has no $x - y$ Hamilton path. (It does have $x - z$ and $y - z$ Hamilton paths, which you may feel free to find, though they won't be useful for the next step.)

   b) Use part (a) to show that the Tutte graph has no Hamilton cycle.

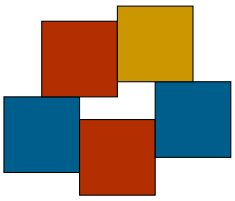
5. Here are several coloring problems for the union of two graphs. The graphs may share vertices and even edges; their union is a graph containing every vertex and every edge present in at least one of the graphs.

   a) The union of two planar graphs is 12-colorable. Explain why this is a special case of Proposition 24.6.

   b) By imitating the proof of Proposition 24.6, prove that the union of two trees is always 4-colorable.

   c) Prove that the union of two bipartite graphs is always 4-colorable.

6. The faraway and fictional continent of Kvadrat is divided into many countries, all in the shape of 1×1 squares. The squares are not necessarily aligned to a grid, and might have unclaimed land between them (which does not need to be given a color). One possible map is shown below:



   Can all possible maps of Kvadrat be colored using only three colors, or is there an example of such a map which requires four colors?

7. The faraway and fictional country of Heibai is divided into cantons, just like Switzerland, but its map has a curious property: at every point where several borders meet, the total number of borders that converge there is even. (The United States has just one point of this type: the point where Arizona, Colorado, New Mexico, and Utah come together.)

   Prove that it's possible to color the cantons of Heibai with just two colors so that two cantons that share a border are always colored differently. (Two cantons that only share a point may have the same color.)

8. Let G be a planar graph that has been partially 5-colored: there is a set $W \subseteq V(G)$ such that every vertex $x \in W$ has already been given a color $f(x) \in \{1, 2, 3, 4, 5\}$, which cannot be changed.

   Prove that if the distance between any two vertices in W is at least 4, then this partial coloring can be completed to a 5-coloring of G. (This result was proved by Michael Albertson in 1998 [1].)

Connectivity

25 Cut vertices

The purpose of this chapter

With this chapter, we embark on the topic of connectivity: exploring how many vertices or edges must be deleted from a graph to make it disconnected, or disconnected in some specific way.

This is the final part of the textbook, so I will spend more time pointing out the connections between the new topics covered in each chapter and other parts of graph theory. (For this chapter, none of the previous topics are critical for understanding; they only serve as examples. Later on, this will change.)

As a graph theorist, I naturally have a graph-theoretic explanation of why pointing out these connections is important. I think of mathematical concepts as forming a very large graph of ideas in my head, with edges representing these connections. This is an undirected graph; in this textbook, I go through some of its vertices in a specific order, but if two ideas are related, either one helps understand the other.

Sometimes you forget things, and the graph of ideas loses a vertex. This is natural. But you want your graph to be resilient to forgetting things. If you lose a vertex with many neighbors, then you can easily recover: you can remember the adjacent ideas and how they related to what you forgot. Don't let your graph of ideas have cut vertices!

25.1 Counting paths

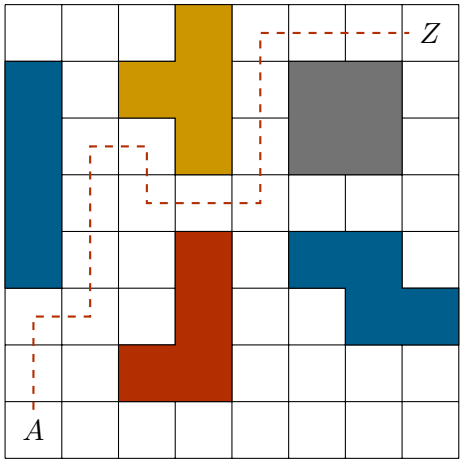
Let me begin with a puzzle.

Problem 25.1. *In Figure 25.1a, several Tetris pieces have been placed on an 8×8 grid. How many ways are there to get from point A (the bottom left corner) to point Z (the top right corner) by a path through the square in the grid that does not pass through any Tetris pieces and does not visit a square more than once?*

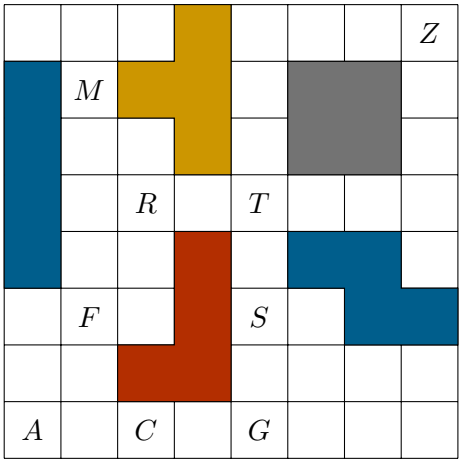
This is a book, so I can't ask you to think about the puzzle on your own for a bit before I reveal the answer. However, I really do think it's a good idea for you to think about it: you'll internalize the ideas in this chapter better if you come up with some of them on your own first. You should especially try to think of a way to solve the problem systematically and with as little brute force as possible: there's many paths, so you don't want to just list them all.

When you've done all the thinking you want to do, keep reading.

No rush—take your time.



(a) The puzzle, with one sample path



(b) A few useful squares

Figure 25.1: Count the paths from A to Z!

Alright! So: the first insight that cuts down on the work required is that the square marked T in Figure 25.1b is special. All paths from A to Z have to go through T (and they are only allowed to go through T once). So every $A-Z$ path is the union of an $A-T$ path and a $T-Z$ path; conversely, the union of every $A-T$ path and every $T-Z$ path is an $A-Z$ path. This means that to get our final count, we can multiply the number of paths from A to T by the number of paths from T to Z.

Question:	Which graph are all these paths in?
Answer:	The graph obtained from the 8×8 grid graph by deleting every vertex covered by a Tetris piece. Squares like A, T, and Z are vertices of this graph.

Let's make up some notation, just for this problem: let $p(x, y)$ be the number of paths from x to y. In this notation, we are looking for $p(A, Z)$, but we've just discovered that $p(A, Z) = p(A, T) \cdot p(T, Z)$.

Question:	What is $p(T, Z)$?
Answer:	It is just 2: from T, we can go up and then right, or right and then up, but the grey square Tetris piece stops us from doing anything else.

Next, we can simplify $p(A, T)$ with just a bit of casework. We can arrive at T either from the left (through the square marked R) or from below (through the square marked S). So we can count both kinds of paths separately.

Question:	Is it true that $p(A, T) = p(A, R) + p(A, S)$?
Answer:	No: as we've defined it, $p(A, R)$ counts all paths from A to R, including ones that go through T first—and we can't follow those up by going back to T.

Instead, let's define a second function $q(x, y)$, to be the number of paths from x to y that don't go through T. Now it's okay to say that $p(A, T) = q(A, R) + q(A, S)$. Let's compute $q(A, R)$ first, and come back to look at $q(A, S)$ second.

Without going through T, every path from A to R must pass through F first: like our initial identity, this lets us factor $q(A, R)$ as $q(A, F) \cdot q(F, R)$. What's more, the paths cannot go through squares C or M: if they do, there's no way to return! So $q(A, F)$ can be computed just by looking at a 3×2 rectangle in the bottom left corner, and $q(F, R)$ can be computed just by looking at a 4×2 rectangle. I have no more tricks to suggest here, but there's not many paths, so we can just count them: $q(A, F) = 4$ and $q(F, R) = 6$.

When we move on to $q(A, S)$, the role of certain squares is reversed. The path cannot go through F, because that cuts off our path of retreat. It must go through C, and then it must go through G. (There is only one way to get from C to G at this point.) So we can factor $q(A, S)$ as $q(A, C) \cdot q(G, S)$. Once again, these are small problems we can solve in just one corner of the grid by listing out all the paths; you should get $q(A, C) = 3$ and $q(G, S) = 8$.

We are now ready to combine the answers we got:

$$\begin{aligned} p(A, Z) &= p(A, T) \cdot p(T, Z) \\ &= \Big(q(A, R) + q(A, S)\Big) \cdot p(T, Z) \\ &= \Big(q(A, F) \cdot q(F, R) + q(A, C) \cdot q(G, S)\Big) \cdot p(T, Z) \\ &= \Big(4 \cdot 6 + 3 \cdot 8\Big) \cdot 2 = 96. \end{aligned}$$

25.2 Cut vertices

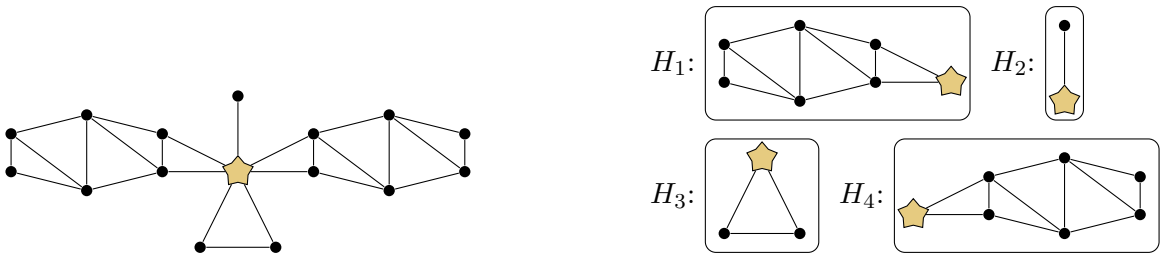
The graph-theoretic concept at work in our solution to Problem 25.1 is the notion of a cut vertex.

A **cut vertex** x of a graph G is a vertex such that $G - x$ has more connected components than G. Most commonly, G is a connected graph, in which case x is a cut vertex if and only if $G - x$ is not connected.

For example, in the graph representing Problem 25.1, vertex T is a cut vertex: deleting it separates A from Z. (Several other vertices marked in Figure 25.1b, such as F and G, are cut vertices of subgraphs of this graph.)

The definition of a cut vertex might resemble the definition of a bridge given in Chapter 9: we defined a bridge to be an edge that disconnects a graph (or increases the number of connected components) when we remove it. For this reason, bridges are sometimes called cut edges.

Question:	We know that in a tree, every edge is a bridge. Which vertices in a tree are cut vertices?
Answer:	The cut vertices are exactly the vertices which are not leaves. When a leaf is deleted, a smaller tree is left; however, deleting a vertex of degree $k > 1$ leaves k connected components, one for every neighbor of the deleted vertex.



(a) A cut vertex in G

(b) H_1, H_2, H_3, H_4

Figure 25.2: Using a cut vertex to break G down into $H_1, H_2, \ldots, H_k$

What are cut vertices good for? It's hard to give a complete answer to a question like that in graph theory, because an easy-to-define concept can pop up in seemingly unrelated questions. On their own, cut vertices are good in applications because they tell us about the reliability of computer networks [53] or supply networks [22], just to name two examples. Usually, identifying the cut vertices does not tell the complete story; in this part of the textbook, you will learn some more complicated measures of reliability, as well.

It can also be helpful to know when a graph has a cut vertex no matter which problem we're trying to solve, because it can help us break down the problem into smaller and simpler problems: we saw that in action in Problem 25.1. Let's be a bit more specific about this, though. Given a connected graph G, suppose that x is a cut vertex of G. Where do we find the subproblems?

Well, we could start by identifying $G_1, G_2, \ldots, G_k$, the connected components of $G - x$. But that's not usually quite what we want, because none of these connected components "remember" how they fit together. Instead, we want subgraphs $H_1, H_2, \ldots, H_k$ that all come together at x. Formally, let H_i be the subgraph of G induced by $V(G_i) \cup \{x\}$; this subgraph includes only one connected component of $G - x$, but also keeps track of how that component is attached to x. An example is shown in Figure 25.2.

Let's look at some examples to see how this helps us solve a few of the problems that have been covered so far in this book.

Question: How does determining whether $H_1, H_2, \ldots, H_k$ are planar help us decide if G is planar?

Answer: If all k subgraphs are planar, their plane embeddings can be combined by placing them around x like the petals of a rose, so G is also planar; Figure 25.2 is an example of this. If one of the subgraphs is not planar, then G as a whole isn't planar, either.

Question: How does coloring the graphs $H_1, H_2, \ldots, H_k$ help us find a coloring of G?

Answer: Though the colorings are not necessarily compatible "out of the box", we can permute the colors used on each subgraph so that all of them give x the same color. Then, by combining the colorings, we get a coloring of all of G.

In particular, $\chi(G) = \max\{\chi(H_1), \chi(H_2), \ldots, \chi(H_k)\}$.

Question:	How does finding the largest clique in $H_1, H_2, \ldots, H_k$ help us find the largest clique in G?
Answer:	A clique in G must be entirely contained in some H_i: if we take two vertices other than x from different subgraphs, then they won't be adjacent. In particular, $\omega(G) = \max\{\omega(H_1), \omega(H_2), \ldots, \omega(H_k)\}$.

Question:	How does finding a spanning tree of $H_1, H_2, \ldots, H_k$ help us find a spanning tree of G?
Answer:	We can just take the union of the spanning trees of all the subgraphs. This is always a spanning tree of G, and every spanning tree of G is obtained in this way, so this can be used to count the spanning trees of G, or to find a minimum-cost spanning tree.

There are many more examples, but let's move on.

25.3 2-connected graphs

What happens if we split up a graph at its cut vertices (as in the previous section) as many times as possible? Eventually, we arrive at a bunch of smaller graphs with no more cut vertices.

We'll probably have to use some other tools to solve whatever problem we were trying to solve in those small graphs. However, maybe the property of having no cut vertices will make them special somehow? We should study graphs with no cut vertices to see if we can learn anything useful about them.

(From the point of view of reliability of a network, graphs with no cut vertices are also special: they are the graphs that stay connected even if something happens to one of the vertices.)

I'm going to do something a bit strange with the definition here, and say that a 2-*connected* graph is a connected graph with at least 3 vertices and no cut vertices.[34] Why is that clause "at least 3 vertices" there?

It's possible to have a 2-vertex graph with no cut vertices: the graph K_2. (We even saw K_2 among the smaller graphs we got in Figure 25.2.) However, in many ways, K_2 is unusual for graphs with no cut vertices. The reason is that we can rephrase the definition as saying "For any three different vertices x, y, and z, if x is deleted, there is still a path from y to z." A graph that doesn't have three different vertices satisfies this definition in a trivial way.

Many of the theorems we prove about 2-connected graphs will be false for K_2, which is why we exclude it from the definition. This starts with the following theorem, which would be false if K_2 were considered to be a 2-connected graph:

[34] I am also making this an italicized, "minor" definition, because in Chapter 26, we will define what it means for a graph to be *k-connected*, and will not need to refer to this special case again.

Theorem 25.1. *A graph G is 2-connected if and only if any two vertices of G lie on a common cycle. (That is, for all $x, y \in V(G)$ there exists a cycle in G through both x and y.)*

Theorem 25.1 is an if-and-only-if result, so it is saying two things:

1. If any two vertices of G lie on a common cycle, then G is 2-connected.
2. If G is 2-connected, then any two vertices of G lie on a common cycle.

Of these, statement 1 is the easier of the two directions to show, so we will go ahead and prove it right away; we will return and prove the second direction of the theorem later in this chapter.

Proof of half of Theorem 25.1. Suppose that any two vertices of a graph G lie on a common cycle. (In particular, G must contain at least one cycle, which means that there must be at least 3 vertices.)

Let x be any vertex of G. To verify that $G - x$ is connected, let y and z be two vertices of $G - x$. How do we show that in $G - x$, there is a $y - z$ path?

Well, in G, there is a cycle containing both y and z. We can represent this cycle by a closed walk

$$(x_0, x_1, \ldots, x_{l-1}, x_0)$$

where $x_0 = y$ and $x_i = z$ for some i. This walk can be split in two:

$$(x_0, x_1, \ldots, x_{i-1}, x_i) \quad \text{and} \quad (x_0, x_{l-1}, \ldots, x_{i+1}, x_i).$$

Both of these walks represent $y - z$ paths, and they have no vertices other than y and z in common. Therefore x (the vertex we delete) can only appear as a vertex on one of the paths, which means that the other $y - z$ path survives to $G - x$. □

Through Theorem 25.1, 2-connected graphs have some ties to topics covered earlier in this book. Here's a quick example:

Proposition 25.2. *If G is a Hamiltonian graph, then it is 2-connected.*

Proof. If G has a Hamilton cycle, then any two vertices of G lie on that cycle, so G is 2-connected by Theorem 25.1. □

Question:	In fact, Proposition 25.2 is a special case of an earlier result about Hamiltonian graphs. How?
Answer:	From Corollary 17.3, we know that all Hamiltonian graphs are tough: if $k \geq 1$ vertices are deleted, at most k connected components are left. In the special case $k = 1$, this tells us that there cannot be any cut vertices.

Much of the early work on 2-connected graphs and connectivity in general was done by Hassler Whitney, who proved Theorem 25.1, Theorem 25.4, Proposition 25.5, and Theorem 26.3 in 1932 [105, 106].

25.4 Ear decompositions

Many times in this textbook, we've discussed the following kind of question: if you've solved a graph-theoretic problem, how do you give a short but convincing demonstration that your solution is correct? Let's recap a few instances of this:

- If two graphs are isomorphic, then we can write down an isomorphism, and it will be tedious but straightforward to check that it is an isomorphism.

  If two graphs are not isomorphic, a quick way to demonstrate this is by finding a graph invariant that the two graphs disagree in. However, finding such an invariant might be hard.

- If we want to know the matching number $\alpha'(G)$ of a bipartite graph, and we've found a large matching M, that's only a proof that $\alpha'(G) \geq |E(M)|$. What if there's a larger matching? But by Kőnig's theorem (Theorem 14.2), we can always find a vertex cover U with $|E(M)| = |U|$, and use it as a proof that M is as large as possible.

- If a graph is Hamiltonian, we might still have to work very hard to find a Hamilton cycle; but once we've found it, it is very easy to check.

  What do we do if a graph G is not Hamiltonian? Sometimes, if we're lucky, the graph will not be *tough* (as defined in Chapter 17), either. In that case, after yet more hard work, we can find a set $S \subseteq V(G)$ such that $G - S$ has more than $|S|$ connected components, proving that G is not tough and not Hamiltonian. But there are graphs for which this will not work; we might not always have a short proof.

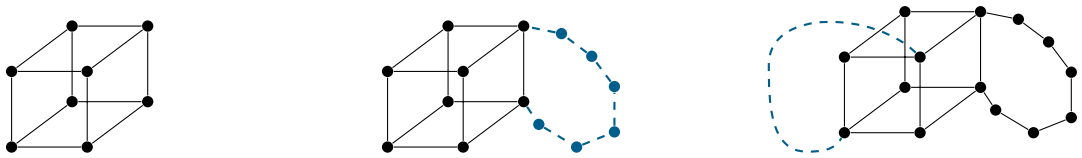
- If a graph is planar, then we can demonstrate this by drawing a plane embedding. What if the graph is not planar? Kuratowski's theorem (Theorem 22.7) tells us that we can demonstrate this by finding a subdivision of $K_{3,3}$ or K_5 inside the graph. Finding the subdivision might take some work, but checking it takes much less work; it's great for busy teachers grading graph theory homework.

These ideas are not just important when applied to individual graphs. They also often come up when writing a proof! Often, proving that something does exist is a lot easier than proving that something does not exist. The ideas here tell us how we can turn the second kind of proof-writing back into the first kind.

Let's think about how the idea of short demonstrations applies to 2-connected graphs.

Question:	Suppose a connected graph G is not 2-connected. How can you demonstrate this?
Answer:	You could find a cut vertex x.

Question:	Okay, but how do you demonstrate that x is a cut vertex?
Answer:	You'd need to show that $G - x$ is not connected; Lemma 3.2 can be a useful tool for this.



(a) The cube graph (b) The cube, plus an ear (c) The cube, plus two ears

Figure 25.3: Adding ears to a cube graph

It's less straightforward to consider the other direction: if a graph is 2-connected, and you want to prove it, how do you do so? Working directly from the definition, you could check for every vertex x that $G - x$ is connected, perhaps by giving a spanning tree of $G - x$ (but verifying that a graph is a tree also takes some work). We could also try to find a collection of cycles so that every pair of vertices lies on a common cycle in our collection, and apply Theorem 25.1. However, we might need a lot of cycles to make this happen, and checking them will not be easy.

All this is a lengthy introduction to a new idea: ears, and ear decompositions.

Let an **ear** of a graph G be a path in G in which every vertex except the first and last has degree 2. From now on, we will also refer to the vertices of a path except the first and last as the **internal** vertices of the path, to make them easier to refer to.

When we **add an ear** to a graph G, we pick two different vertices x_0 and x_l already in $V(G)$; we add new vertices $x_1, x_2, \ldots, x_{l-1}$ and new edges $x_0x_1, x_1x_2, \ldots, x_{l-1}x_l$. This creates the ear represented by the walk $(x_0, x_1, x_2, \ldots, x_l)$.

It's easier to show what it means to add an ear by means of a picture, rather than with words. Figure 25.3 shows how, starting with a cube graph (Figure 25.3a), we add an ear to arrive at Figure 25.3b. This particular ear has many internal vertices, but that's not necessary. Adding an edge to a graph (as in Figure 25.3c) is a special case of adding an ear: it's adding an ear of length 1.

The following lemma is the reason why we introduce ears.

Lemma 25.3. *If G is a 2-connected graph and we add an ear to G, the resulting graph is also 2-connected.*

Proof. Let H be a graph obtained from G by adding an ear R: an $x - y$ path for two different vertices $x, y \in V(G)$ whose internal vertices only exist in H. We will prove that the new graph H still does not have a cut vertex.

To do this, we see what happens when we delete a vertex of H:

- Suppose we delete a vertex $z \in V(G)$ other than x or y. Because $G - z$ is connected, all vertices of $G - z$ are in the same connected component of $H - z$. Also, vertices of R all have a path to x and y (because R is connected), so they are also in that same connected component: $H - z$ is connected.

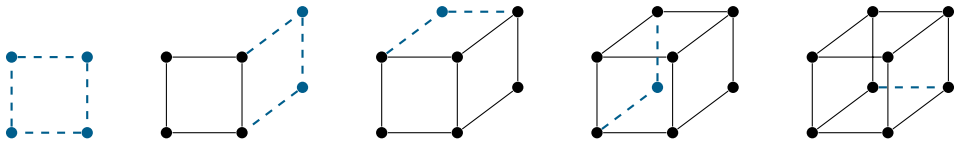


Figure 25.4: An ear decomposition of the cube graph

- If we delete x, essentially the same thing happens. The only change is that an internal vertex of the ear we added to G has only only one path to the connected component of $G - x$: its path to y in $R - x$. Swapping the roles of x and y, the same thing happens if we delete y.
- If we delete a vertex z which is an internal vertex of R, then G remains connected, and $R - z$ is divided into two path components: one containing x and one containing y. Every internal vertex of R other than z still has a path to either x or y, so it is in the same connected component as the vertices of G.

In all cases, H remains connected when we delete a vertex, so it is 2-connected. □

In particular, if we start with a cycle and repeatedly add ears, the result will be connected by Lemma 25.3 and by induction. We can use this to give a short proof that a graph G is 2-connected, if we can find a way to build G from a cycle inside G by adding ears. Figure 25.4 shows a way to do this for the cube graph Q_3.

We refer to a demonstration of 2-connectedness via Lemma 25.3 as an ear decomposition. Since we'd like to use ear decompositions in proofs, let's give a formal definition. An **ear decomposition** of a graph G is a sequence $R_1, R_2, \ldots, R_k$ of subgraphs of G with the following properties:

1. R_1 is a cycle.
2. R_i is an ear of $R_1 \cup R_2 \cup \cdots \cup R_i$ for all $i > 1$.
3. Each edge of G is in exactly one of the subgraphs R_i.

   As a consequence, $G = R_1 \cup R_2 \cup \cdots \cup R_k$.

(The last of these three properties is, as you might remember, exactly what we mean by a decomposition in general.)

Does such a proof always exist? Yes!

Theorem 25.4. *A graph G is 2-connected if and only if it has an ear decomposition.*

Proof. In both directions of this proof, we will have a sequence $R_1, R_2, \ldots$ of subgraphs of G, which we will either assume to be an ear decomposition or prove to be all or part of one. To simplify notation, let's write G_i for the union $R_1 \cup R_2 \cup \cdots \cup R_i$. In the case of an ear decomposition, R_i should be an ear of G_i, and we obtain G_{i+1} from G_i by adding the ear R_{i+1}.

If G has an ear decomposition, then it is 2-connected by repeated use of Lemma 25.3 in a short induction. Let the sequence $R_1, R_2, \ldots, R_k$ be the ear decomposition; then we prove by induction on i that G_i is 2-connected. The base case $i = 1$ holds because cycles are 2-connected:

deleting any vertex from R_1 leaves a path. Since G_{i+1} is obtained from G_i by adding an ear, Lemma 25.3 is exactly the induction step we need. Finally, $G_k = G$, so reaching $i = k$ proves that G is 2-connected.

To prove the other direction of the if-and-only-if statement, we have to reason in the opposite way from Lemma 25.3, building up the ear decomposition $R_1, R_2, \ldots, R_k$ step by step.

We can start by letting R_1 be any cycle in G. Why does a cycle exist? Well, we know G is connected and has at least three vertices. If G contained a leaf x, then the only neighbor of x would be a cut vertex, separating x from the rest of the graph. This is not allowed, so G must have minimum degree at least 2, giving it a cycle by Theorem 4.4.

Question:	Doesn't this also follow from Theorem 25.1?
Answer:	It does, but I want to avoid that argument. We are soon going to use ear decompositions to prove Theorem 25.1, and I want to make sure that proof is not circular.

Next, suppose we've gotten partway through the process, finding the subgraphs $R_1, R_2, \ldots, R_i$. These should satisfy the first two properties of an ear decomposition. Because we're not done yet, they don't have to satisfy the third property. However, we assume that $R_1, R_2, \ldots, R_i$ share no edges, and that $G_i = R_1 \cup R_2 \cup \cdots \cup R_i$ is a subgraph of G. We would like to make G_i bigger by adding an ear, which we'll call R_{i+1}; however, we want to add an ear that's still entirely contained in G.

The first question we ask is this: is $V(G_i) = V(G)$? If so, then we're nearly done, and continuing the ear decomposition is easy. Pick any edge $xy \in E(G)$ that is not in G_i, and make that edge (or, more precisely, the subgraph consisting of x, y, and edge xy) be the ear R_{i+1} we add next.

Question:	What will steps $i+2, i+3, \ldots$ look like, in this case?
Answer:	Since $V(G_{i+1}) = V(G)$ as well, we'll stay in this case until the end. We will add the remaining edges of G, one at a time, as ears of length 1.

So it's the early steps that we have to worry about, when $V(G_i)$ is still not all of $V(G)$. In this case, let xy be any edge where $x \in V(G_i)$ and $y \notin V(G_i)$; we will try to construct an ear that begins with the edge xy.

Question:	How do we know such an edge xy exists?
Answer:	If there is no such edge, then G would not have any edges between $V(G_i)$ and its complement $V(G) - V(G_i)$, so it would not be connected.

Since G is 2-connected, in particular $G - x$ is connected, so we can find a path in $G - x$ from y to any other vertex. Choose that other vertex to be a vertex $z \in V(G_i)$, not equal to x, of course. There is a $y - z$ path in $G - x$; represent it by a walk $(x_1, x_2, \ldots, x_l)$ with $y = x_1$ and $z = x_l$. By setting $x_0 = x$, we get an walk $(x_0, x_1, x_2, \ldots, x_l)$; this walk represents an $x - z$ path that starts and ends in G_i.

Question:	Is this path an ear we can add to G_i?
Answer:	Not necessarily.

Question:	What could be the matter with it?
Answer:	We don't know for sure that none of its internal vertices are in $V(G_i)$. We only know this about x_1, because $x_1 = y$ was chosen for this purpose.

To fix this, let $j > 0$ be the first positive number such that $x_j \in V(G_i)$. (Since $x_l = z$ and $z \in V(G_i)$, we know that j exists, but it's possible that we return to $V(G_i)$ before reaching z; in that case, $j < k$.) Now the truncated walk $(x_0, x_1, x_2, \ldots, x_j)$ represents an ear we can add to G_i: if we define R_{i+1} to be the path represented by this walk, then R_{i+1} is an ear of $R_1 \cup R_2 \cup \cdots \cup R_{i+1}$.

Question:	How do we know this?
Answer:	All the internal vertices of R_{i+1} are not in G_i, so in $G_i \cup R_{i+1}$, they only have two neighbors: their neighbors along the path R_{i+1}. This also shows that the edges R_{i+1} are not in G_i, because they all involve at least one of these internal vertices.

To recap, we've shown that in all cases, if we've picked $R_1, R_2, \ldots, R_i$, then we can continue by picking R_{i+1} that can be part of the ear decomposition. At each step, we add at least one edge that wasn't in the partial ear decomposition yet; therefore we're guaranteed to eventually build all of G, and obtain an ear decomposition of G. □

An important observation is that no matter how we've constructed $R_1, R_2, \ldots, R_i$, if we haven't finished, then we can always choose R_{i+1} somehow. This makes the algorithm in the proof a greedy one! The initial cycle R_1 can be any cycle, and later on, each ear R_i can be any ear. We don't have to worry about making choices that lead us to a dead end later.

In particular, the strategy of picking vertices x, y, z, finding a $y - z$ path, and seeing where it returns to $V(G_i)$ is just one way to guarantee that we find an ear. We do not have to follow this strategy if have an alternative. If we're working with a small graph, such as the cube graph whose ear decomposition we found in Figure 25.4, then it's much easier to just look at the graph and find an ear among the vertices and edges we haven't included yet.

Question:	Is there a shorter ear decomposition of the cube graph?
Answer:	Yes and no. We can work to get to a spanning subgraph more quickly, at which point we just have edges to add; for example, we could let R_1 be a Hamilton cycle in the cube graph. However, we'll have to add those edges one at a time, and we'll still have the same total number of steps.

In a practice problem at the end of this chapter, I'll ask you to prove that this is always true: two ear decompositions of the same graph always have the same number of pieces.

25.5 Induction on ears

Theorem 25.4 can be thought of as an inductive definition of 2-connected graphs: a graph G is 2-connected if and only if it is either a cycle, or the result of adding an ear to a smaller 2-connected graph. Whenever we have such a definition of a combinatorial object of any kind, it suggests that induction is a good strategy for proving properties of such an object. (See Appendix B for more details about such definitions.)

Let me begin by giving you an example, which is an application of 2-connectedness to planar graphs. (I will be lighter on the geometric details, because they are not the focus here. In a more careful approach, using Lemma 21.2 to confirm that all edges must separate two faces is a key step.)

Proposition 25.5. *If a planar graph G is 2-connected, then in every plane embedding of G, the boundary of every face is a single cycle.*

Question:	What are all the ways in which the boundary of a face might fail to be a cycle?
Answer:	It's possible that a face has multiple boundary walks. This happens when G is not connected, and the face touches multiple components of G. It's possible that a boundary walk does not represent a cycle because repeats an edge; we know that this happens when the edge is a bridge. Finally, a boundary walk might not repeat any edges, but still repeat a vertex x. In this case, x is a cut vertex; intuitively, because we can draw a closed curve in the face that only crosses the embedding at x, and separates two parts of G. In a plane embedding of $G - x$, the same closed curve will separate two connected components.

Proof of Proposition 25.5. Since G is planar, begin by picking a plane embedding of G.

Since G is 2-connected, let $R_1, R_2, \ldots, R_k$ be an ear decomposition of G (which exists by Theorem 25.4) and let $G_i = R_1 \cup R_2 \cup \cdots \cup R_i$. Since G_i is a subgraph of G, the plane embedding of G contains a plane embedding of G_i: just erase every vertex and edge of G that's not in G_i.

We will prove by induction on i that in this plane embedding of G_i, the boundary of every face is a single cycle. The base case is $i = 1$. In this case, $G_1 = R_1$ is a cycle ; in every plane embedding of the cycle, there are two faces, an inner face and an outer face, and the entire cycle is the boundary between them.

Now assume for some $i > 1$ that in the plane embedding of G_{i-1}, the boundary of every face is a single cycle. Then G_i is obtained from G_{i-1} by adding the ear R_i: a path whose ends are in G_{i-1}, but whose internal vertices are not. In the plane embedding, this path cannot cross any

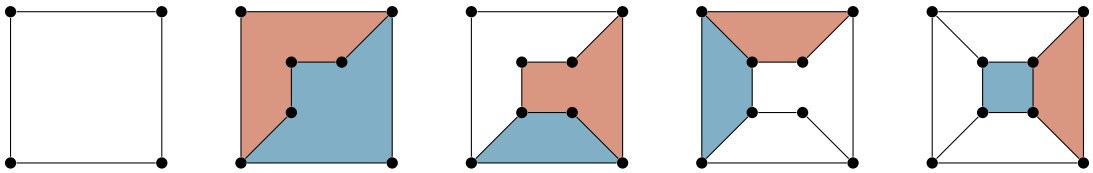


Figure 25.5: In a plane embedding of the cube graph, each ear divides a face in two

edges of G_{i-1}, so there must be some face F in G_{i-1} that contains it entirely. The boundary of F in G_{i-1} is a cycle, by the induction hypothesis.

In G_i, F is divided by R_i into two faces $F_1 \cup F_2$. Their boundaries must share the path R_i, and divide up the original boundary of F somehow. The only way to do this is by two cycles, each of which follows the boundary of F from one endpoint of R_i to the other, and returns along R_i. See Figure 25.5 for an illustration of such induction steps in a planar ear decomposition of the cube graph; in each diagram, the two faces divided by the recently added ear are shaded.

This completes the induction. Now, Proposition 25.5 follows from the $i = k$ case, because G_k is all of G. □

In general, inducting on ear decompositions follows the pattern of this proof. First, we choose an ear decomposition, and verify that the claim holds for R_1 (a cycle). Next, we verify that if the claim is true for G_{i-1}, it holds for G_i: in other words, that the claim continues to hold if we add an ear.

Earlier, we observed that an ear decomposition can be found greedily; in particular, that we can begin the ear decomposition by choosing R_1 to be any cycle we like. This is occasionally useful when writing a proof by induction, since we can make sure that some key vertices end up in R_1.

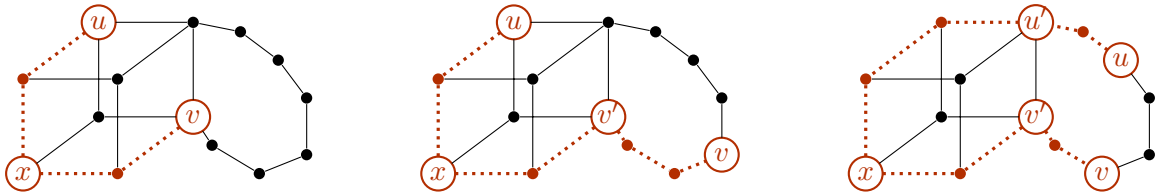
Of course, if we want to do this, we need to prove that the cycle we want exists, first. Once we've shown Theorem 25.1, it will be an excellent tool for the base case, because it's all about cycles existing. Unfortunately, my plan for Theorem 25.1 is to use an induction on ear decompositions! So we will need to begin with a silly lemma that will no longer be necessary once Theorem 25.1 is there to replace it, but will be useful in that particular proof.

Lemma 25.6. *If G is 2-connected, and x is any vertex, then G has a cycle containing x.*

Proof. To have any hope of finding a cycle through x, we need to know that x has two neighbors. Well, if x had only one neighbor, we'd be able to delete that neighbor to disconnect x from the rest of the graph: the neighbor of x would be a cut vertex.

So we can choose two neighbors of x; call them u and v.

Because G is 2-connected, $G - x$ is still connected, and in particular there is a $u - v$ path P in $G - x$. We obtain a cycle through x from P by adding vertex x and edges $\{ux, vx\}$ to P. □



(a) Case 1 (b) Case 2 (c) Case 3

Figure 25.6: Three cases in the induction step of Lemma 25.7

Question:	Theorem 25.1 would be false if we considered K_2 to be a 2-connected graph, and so would Lemma 25.6. At which step would the proof fail, if $G = K_2$?
Answer:	When we talk about a vertex deletion that would "disconnect x from the rest of the graph", we assume that there is a rest of the graph, which is false in the case of K_2.

In the next section, we will see two more examples of induction on ear decompositions, and we will see how Lemma 25.6 fits in.

25.6 Finding common cycles

Before we prove Theorem 25.4, we will prove a second lemma. Unlike Lemma 25.6, this one remains useful in the study of 2-connected graphs; it's not just a tool for this one particular proof.

Lemma 25.7. *If G is 2-connected, and u, v, x are any three vertices, then G contains a $u - v$ path that passes through x.*

Proof. By Lemma 25.6, G contains a cycle containing x. Let $R_1, R_2, \ldots, R_k$ be an ear decomposition of G in which R_1 is a cycle containing x, and let $G_i = R_1 \cup R_2 \cup \cdots \cup R_i$. We will prove by induction on i that if u and v are any two vertices in G_i, then G_i contains a $u - v$ path that passes through x.

When $k = 1$, the graph G_1 consists only of the ear R_1: it is a cycle. No matter where u and v are on the cycle, there are two $u - v$ paths going around the cycle from u in either direction, and one of them passes through x, proving the base case.

Now assume that for some $i > 1$ that the paths we want exist in G_{i-1}. Then G_i is obtained from G_{i-1} by adding the ear R_i. Let u and v be arbitrary vertices of G_i; we consider three cases for where they might be, illustrated in Figure 25.6.

Case 1. Both u and v are vertices of G_{i-1}. By the induction hypothesis, G_{i-1} contains a $u - v$ path through x. This is also a path in G_i.

Case 2. Only one of u and v is a vertex of G_{i-1}; without loss of generality, $u \in V(G_{i-1})$ and v is an internal vertex of R_i. In this case, let u' be the start or end of R_i. By the induction

hypothesis, G_{i-1} contains a $u' - v$ path through x. We can extend it to a $u - v$ path through x in G_i if we begin by going from u to u' along the ear R_i.

Case 3. Both u and v are internal vertices of R_i. In this case, let u' and v' be the ends of R_i closer to u and to v, respectively; that is, R_i is a path of the form $(u', \dots, u, \dots, v, \dots, v')$. By the induction hypothesis, G_{i-1} contains a $u' - v'$ path through x. We can extend it to a $u - v$ path through x in G_i if we begin by going from u to u' and end by going from v to v', both along the ear R_i.

In all cases, we've found the $u - v$ path through x we wanted; therefore G_i contains such a path for any $u, v \in V(G_i)$. By induction, this is true for all i and in particular for $i = k$; because $G_k = G$, this proves the lemma. □

With the aid of Lemma 25.3, a second induction on the ear decomposition is now enough to prove Theorem 25.1 that any two vertices x and y of a 2-connected graph G lie on a common cycle.

Proof of Theorem 25.1. Let x and y be two vertices of a 2-connected graph G; we want to show that there is a cycle in G containing both x and y.

By Lemma 25.6, G contains a cycle containing x. Let $R_1, R_2, \dots, R_k$ be an ear decomposition of G in which R_1 is a cycle containing x, and let $G_i = R_1 \cup R_2 \cup \cdots \cup R_i$. We will prove by induction on i that if $y \in V(G_i)$, then G_i contains a cycle that passes through both x and y.

When $i = 1$, this is automatic: $G_1 = R_1$ is the cycle we want.

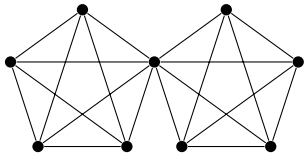
Now assume for some $i > 1$ that every vertex $y \in V(G_{i-1})$ lies on a cycle in G_{i-1} that also passes through x. We would like to prove the same thing for G_i, which is obtained from G_{i-1} by adding the ear R_i. Since we have the induction hypothesis, we only need to find a cycle through x and y when y is one of the new vertices: an internal vertex of R_i.

Let u and v be the ends of ear R_i. By Lemma 25.6, there is a $u - v$ path P in G_{i-1} passing through x. The paths P and R_i have only their ends u and v in common: none of the internal vertices of R_i are in $V(G_{i-1})$, and all vertices of P are. Therefore the union $P \cup R_i$ is a cycle containing x (because P contains x) and y (because R_i contains y).

By induction, for every $y \in V(G_k)$ there is a cycle in G_k that passes through x and y; since $G = G_k$, this proves the theorem. □

25.7 Practice problems

1. Give an example of each of the following:
   a) A 10-vertex graph with 8 cut vertices.
   b) A graph with the same number of vertices and edges as in (a), but only one cut vertex.
   c) A regular graph with only one cut vertex.
2. Let G be a graph consisting of two copies of K_5 joined at a vertex:



How many spanning trees does G have?

3. Suppose G has a cut vertex x, and that the graphs $H_1, H_2, \ldots, H_k$ are the subgraphs of G meeting at x (as in Figure 25.2).

   a) Explain why knowing the independence numbers $\alpha(H_1), \alpha(H_2), \ldots, \alpha(H_k)$ is not enough to find $\alpha(G)$.

   b) Without knowing G, if all I tell you is the values of $\alpha(H_1), \alpha(H_2), \ldots, \alpha(H_k)$, what are the minimum and maximum possible values of $\alpha(G)$?

4. Prove that if G is 2-connected, and H is a *subdivision* of G (as defined in Chapter 22), then H is 2-connected.

5. Find an ear decomposition of:

   a) $K_{3,3}$.

   b) $K_{2,5}$.

   c) $K_{2,n}$ for every n.

6. You might wonder: is there a shortest ear decomposition in a graph? In fact, every ear decomposition contains the same number of pieces.

   a) Suppose that G has the ear decomposition $R_1, R_2, \ldots, R_k$, where R_1 is a cycle of length l_1 and for each $i \geq 2$, R_i is a path of length l_i.

   Find the number of edges in G in terms of $l_1, l_2, \ldots, l_k$.

   b) Find the number of vertices in G in terms of $l_1, l_2, \ldots, l_k$.

   c) If G has n vertices, m edges, and k ears in an ear decomposition, find a relationship between n, m, and k from your answers to the first two parts.

   Conclude that the value of k is predetermined by G, and doesn't depend on the ear decomposition.

7. One way to interpret Theorem 25.1 is that for any two vertices x, y in a 2-connected graph G, there are two $x - y$ paths that share none of their internal vertices. (Such paths are said to be *internally disjoint*, a term we will introduce in Chapter 26.) Here is an INCORRECT proof of a FALSE generalization:

   Claim. If G is 2-connected and P is any $x - y$ path, there is a second $x - y$ path P' that shares no vertices with P other than x and y.

   "Proof". Suppose there is no such path P'. That means that all other paths P' end up sharing some other vertex of P. But then, deleting that vertex of P destroys all $x - y$ paths, and so that vertex was a cut vertex. This cannot happen in a 2-connected graph; contradiction! So a path P' must exist.

a) Point out the mistake in the proof.

b) Give a counterexample to the claim.

8. Let G be a connected graph with at least 3 vertices in which xy is a bridge (that is, $G - xy$ is not connected.)

   a) Prove that either x or y is a cut vertex of G. Give an example showing that they're not necessarily both cut vertices of G.

   b) Prove that xy is a cut vertex of the line graph $L(G)$.

9. In a 3-regular graph:

   a) Prove that a vertex x is a cut vertex if and only if it is incident to a bridge.

   b) Prove that every cut vertex is incident to either 1 or 3 bridges.

   c) Prove that the number of cut vertices is even.

10. The ear decompositions in this chapter are sometimes called proper ear decompositions. There is also a corresponding notion of improper ear decompositions. In the improper case, an ear of a graph G is also allowed[35] to be a cycle in which every vertex except one has degree 2. (That is, when we add an ear, we are allowed to start and end at the same vertex.)

    a) Prove by example that a graph with an improper ear decomposition is not necessarily 2-connected.

    Instead, improper ear decompositions correspond to *2-edge-connected* graphs: connected graphs with no bridges. (These are a special case of a definition we will make in Chapter 26.)

    b) Prove that if a graph G has an improper ear decomposition, then it is 2-edge-connected.

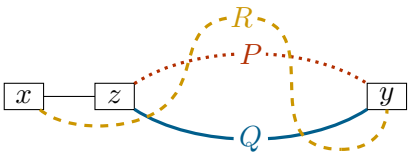
    c) Prove that if a graph is 2-edge-connected, then it has an improper ear decomposition.

11. There is an alternate proof of Theorem 25.1 which does not use ear decompositions. Instead, we prove that any two vertices x, y lie on a common cycle (equivalently, there are two $x - y$ paths with no internal vertices in common) by inducting on the distance $d(x, y)$.

    a) For the base case, prove (without relying on any of our other results) that if G is 2-connected, then for any edge $xy \in E(G)$, there is a cycle containing xy.

    b) For the induction step, we assume that Theorem 25.1 holds for any two vertices at some distance $k \geq 1$, and let x and y be two vertices with $d(x, y) = k + 1$. To apply the induction hypothesis, we let z be the first vertex on a shortest $x - y$ path, so that $d(z, y) = k$. Let P and Q be two internally disjoint $z - y$ paths, and let R be an $x - y$ path in $G - z$, as shown below:

[35] An "improper" ear decomposition should maybe be called a "not-necessarily-proper" ear decomposition: it could be proper, it just doesn't have to be.



Prove that no matter how R intersects P and Q, we can find two $x - y$ paths with no internal vertices in common.

26 Connectivity

The purpose of this chapter

I'm hoping that the idea behind the definitions in this chapter feels intuitive: that to measure the resilience of a graph, it makes sense to see how much you have to remove from the graph to disconnect it. There is a lot more to these definitions, and they turn out to have connections to all parts of graph theory. However, it's going to take a bit of time to get there, and the content of this chapter is mostly setup.

When I've taught this material in the past, I've sometimes focused only on vertex connectivity and not on edge connectivity, for reasons of time. If you do have the time, I think it is nice to look at both, because the contrast helps you understand the logic of each case. For example, if it's hard at first to see why internally disjoint paths are what we need for lower bounds on $\kappa(s,t)$, then edge-disjoint paths and their lower bounds on $\kappa'(s,t)$ provide a second example of the same idea for comparison.

The final section on duality is a topic I could have discussed earlier in the book, but now we have lots of examples to look back to. I should note that in graph theory, duality is a relationship between optimization problems that just sometimes appears; we will occasionally prove it appears, but not explain why. Using linear programming, we could obtain more of an explanation, and even deduce what the correct duals of some optimization problems should be without seeing them before. That would take us too far afield, though.

26.1 Vertex and edge cuts

In the previous chapter, we defined cut vertices and 2-connected graphs; much earlier in the book, we defined bridges. In this chapter, we will discuss generalizations of all three definitions.

We begin with the definition of a **vertex cut** in a graph G. This is subset $U \subseteq V(G)$ such that when all vertices in U are deleted, the remaining graph $G - U$ is not connected.

Don't confuse "vertex cut" and "cut vertex", though they have the same words in a different order! A cut vertex is a vertex, with "cut" as the adjective: it is a vertex which cuts. A vertex cut is a cut (as in, "I sliced the cake in half with a single cut"), with "vertex" as the adjective: it is a cut made up of vertices.

The two terms are related, though. If G is a connected graph, a vertex x is a cut vertex of G if and only if the set $\{x\}$ is a vertex cut of G.

Question: What if G is not connected?

Answer: In this case, cut vertices are those that increase the number of connected components. Meanwhile, $\{x\}$ is almost always a vertex cut; the exception is when x is an isolated vertex and the rest of the graph is connected.

Usually, this will not matter, because we will be looking for the smallest vertex cuts we can. If G is not connected, we will simply say that $\varnothing$ (the empty set) is a vertex cut.

Making vertex cuts an optimization problem in this way makes sense, because it is not surprising when we disconnect a graph by deleting many vertices. We ask: what is the least number of vertices whose deletion disconnects the graph? Unfortunately, before we make that into a definition, we have to tackle an awkward obstacle.

Question: Is it always possible to disconnect a graph by deleting enough vertices?

Answer: Almost always, except for complete graphs!

In K_n, any two vertices are adjacent, and will remain adjacent no matter how many other vertices we remove. This means that (aside from the question of what happens if we delete every single vertex) we can't disconnect K_n by deleting some of its vertices: K_n has no vertex cuts.

This is related to what happened with 2-connected graphs in the previous chapter: we didn't want K_2 to be 2-connected, because most of the theorems about 2-connected graphs do not apply to it, so we specifically excluded it from the definition.

Accordingly, we define the **connectivity** (or **vertex connectivity**) $\kappa(G)$ of a graph G to be the smallest number of vertices in a vertex cut of G, if such a vertex cut exists. If not, then G is isomorphic to K_n for some n, and we "artificially" define $\kappa(K_n) = n - 1$.

We also say that a graph G is ***k*-connected** if $\kappa(G) \geq k$. This is in line with our previous definitions. For graphs with at least 2 vertices, "1-connected" and "connected" are synonyms; meanwhile, 2-connected graphs (as defined in Chapter 25) still have the same definition if we set $k = 2$ in the definition of k-connected graph.

Question: Why does the definition of k-connected graphs have an inequality: $\kappa(G) \geq k$, rather than $\kappa(G) = k$?

Answer: We say "G is k-connected" when we want to say that G is connected enough for something to work. For example, all 2-connected graphs have ear decompositions, even if they are also 3-connected or 100-connected.

This is similar to the way we defined k-colorable graphs: they are the graphs with $\chi(G) \leq k$.

We must specify "vertex cut" and we often specify "vertex connectivity" because all of these definitions have an equivalent for deleting edges, rather than vertices, of a graph. The story is almost the same for edges, so I will just present all the definitions at once.

An **edge cut** in a graph G is a subset $X \subseteq E(G)$ such that when all edges in X are deleted, the remaining graph $G - X$ is not connected.

The **edge connectivity** of G, denoted $\kappa'(G)$, is the smallest number of edges in an edge cut of G, if G has at least two vertices. If G has only one vertex, then we set $\kappa'(G) = 0$.

Finally, a graph G is ***k*-edge-connected** if $\kappa'(G) \geq k$.

Question:	Do we need to take any special care when $G = K_n$?
Answer:	Not unless $n = 1$ (which the definition addresses). When G at least 2 vertices, it is always possible to disconnect G by deleting some edges: in particular, deleting all edges would be enough.

In Chapter 20, we discussed the correspondence between "vertex versions" and "edge versions" of a problem, and used the same notation: for example, $\alpha'(G)$ and $\chi'(G)$ are the edge versions of $\alpha(G)$ and $\chi(G)$. I will use the same notation here. In other sources, $\lambda(G)$ is also a common notation for edge connectivity, but I'd rather not make you memorize even more Greek letters.

Question:	Often the edge version of a problem is just the vertex version, but applied to the line graph $L(G)$ rather than G. Is that true here—is $\kappa'(G) = \kappa(L(G))$?
Answer:	Not quite. I leave the details to practice problem, but a relationship exists only in one direction: $\kappa'(G) \leq \kappa(L(G))$.

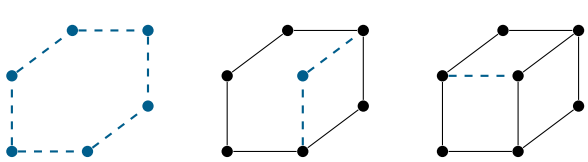
Let X be an edge cut in G, and let S be the set of vertices in one connected component of $G-X$. Then X must contain all edges of G with one endpoint in S and one endpoint in $V(G) - S$. Though X could contain other edges, this is wasteful: just deleting all the edges between S and $V(G) - S$ is enough to disconnect G.

Accordingly, for $S \subseteq V(G)$, we define the **edge boundary** of S to be the set of all edges of G with exactly one endpoint in S. We denote it $\partial_G(S)$ or just $\partial(S)$ when the graph G does not need to be specified.

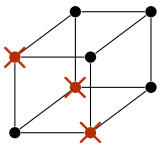
Suppose X is a minimum edge cut, or even a *minimal* one (by the distinction described in Chapter 13): we cannot remove any edges from X and still have an edge cut. In that case, $X = \partial(S)$ for some S. Another way to phrase this is the following proposition:

Proposition 26.1. *Every edge cut X in a connected graph contains an edge boundary $\partial_G(S)$ for some set of vertices S that is neither $\varnothing$ nor all of $V(G)$. In particular, $\kappa'(G)$ can be computed as the minimum size $|\partial_G(S)|$ of all such edge boundaries.*

We do not consider $S = \varnothing$ or $S = V(G)$ because these are the two cases that can never be a connected component of $G - X$, when X is an edge cut.



(a) An ear decomposition of $Q_3 - x$



(b) A vertex cut in Q_3

Figure 26.1: Determining the connectivity of the cube graph

Question:	Is every set of the form $\partial(S)$ a minimal edge cut?
Answer:	No: it's possible that $\partial(S)$ contains a smaller edge cut of the form $\partial(T)$ for a different set T. For an extreme example, consider a bipartite graph with bipartition (A, B). Then the edges $\partial(A)$ we need to separate A from B are all the edges of G; the set $\partial(\{x\})$ for a single vertex x can be much smaller.

26.2 Some examples

In Chapter 25, we saw that the cube graph Q_3 is 2-connected. In fact, we can go one step further.

Proposition 26.2. $\kappa(Q_3) = 3$.

Proof. Ear decompositions are a nice way to prove that graphs are 2-connected, but we don't yet have an equally nice way to show that graphs are 3-connected. We will find one later in this chapter, but for now, let's resort to trickery.

We know Q_3 has no cut vertex; does it have a cut $\{x, y\}$ of size 2? If it did, then in $Q_3 - x$, vertex y would be a cut vertex: deleting it would bring us to $Q_3 - \{x, y\}$, which by assumption is not connected. So we can check for this possibility by checking whether $Q_3 - x$ is 2-connected.

Normally, we'd have to do this by checking 8 possibilities for x. In the case of the cube graph, there are enough symmetries (automorphisms of the cube graph) that it's enough to check one vertex: all 8 cases will be identical.

Figure 26.1a shows an ear decomposition of $Q_3 - x$ for an arbitrary vertex x. (Well, it's only arbitrary mathematically; I deleted the vertex "in the back" of the cube to make the diagram look nicer.) This shows that $Q_3 - x$ is 2-connected, so Q_3 is 3-connected.

Can it be 4-connected? No, because Q_3 has a 3-vertex cut, shown in Figure 26.1b. Therefore $\kappa(Q_3)$ is exactly 3. □

The vertex cut we chose in Figure 26.1b is rather boring; we've disconnected a single vertex from the rest of the graph. Such a vertex cut is possible in every graph G, except in the unfortunate case of (you guessed it) complete graphs. By deleting all edges incident to a vertex x, we also disconnect x from the rest of G, so there is an edge cut of this type as well. To make the cuts as small as possible, we want to choose a vertex x of degree $\delta(G)$: the minimum degree of G.

This is not universal; we wouldn't study connectivity if it always turned out to be equal to the minimum degree. However, the following relationship (also proved by Whitney, like most of the results in Chapter 25) always holds:

Theorem 26.3. *For any graph G, $\kappa(G) \leq \kappa'(G) \leq \delta(G)$.*

Proof. The second inequality, $\kappa'(G) \leq \delta(G)$, follows from our discussion just before the theorem: if x is a vertex of degree $\delta(G)$, then deleting all $\delta(G)$ edges incident to x will disconnect x from the rest of the graph, so it is always an edge cut. (As usual with optimization problems, finding a particular solution gives us an inequality, because there might be better solutions.)

The more difficult inequality is the first. To prove that $\kappa(G) \leq \kappa'(G)$, we need to do the following: given an edge cut X of size $\kappa'(G)$, find a vertex cut U with $|U| \leq |X|$. By Proposition 26.1, we can assume that $X = \partial_G(S)$ for some set S such that $\varnothing \neq S \subsetneq V(G)$.

I like the proof of this inequality because it is an excellent exercise in attempting an argument, identifying the holes in it, and then patching the holes. A few rounds of this, and we will have a complete proof.

Let's begin with the following strategy: to find U, go through the edges in $\partial(S)$, and for each $xy \in \partial(S)$, add either x or y to U. Deleting an endpoint of xy also deletes the edge xy, so after these deletions, S (or what's left of it) is disconnected from the rest of the graph.

Question:	What could go wrong?
Answer:	It's possible that in deleting these endpoints, we've actually deleted every vertex in S, or every vertex not in S, so the remaining graph is still connected.

Okay, so let's make sure that we avoid this. Begin by picking a vertex $s \in S$ and a vertex $t \notin S$. For every edge $xy \in \partial(S)$, add either x or y to U, but with a restriction: if $x = s$, add y to U, and if $y = t$, add x to U. Then in $G - U$, there is no way to get from s to t, because $s \in S$, $t \notin S$, and we have destroyed all edges leaving S.

Question:	What could go wrong?
Answer:	The restriction we've added doesn't leave us an option to choose if one of the edges in $\partial(S)$ is the edge st.

Okay, fair enough; let's make sure that we pick a vertex $s \in S$ and a vertex $t \notin S$ that are not adjacent.

Question:	What could go wrong?
Answer:	Maybe every vertex in S is adjacent to every vertex not in S, so that no such s and t can be picked.

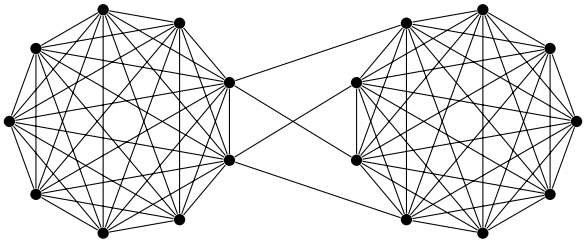


Figure 26.2: A graph with $\kappa(G) = 2$, $\kappa'(G) = 4$, and $\delta(G) = 8$

This is a surprising situation to be in! Let G have n vertices, and let $|S| = k$. Then $\partial(S)$ contains all edges between the k vertices in S and the $n - k$ vertices not in S: this is a lot: $|\partial(S)| = k(n-k)$. We have assumed that $\kappa'(G) = |\partial(S)| = k(n-k)$, but in fact, $k(n-k) > n-1$ except when $k = 1$ or $k = n - 1$.

Since we already know that $\kappa'(G) \leq \delta(G)$, and in every n-vertex graph, $\delta(G) \leq n - 1$, only one possibility remains: $\kappa'(G) = n - 1 = \delta(G)$.

Question:	What graph must G be?
Answer:	G must be isomorphic to K_n; this is the only n-vertex graph with minimum degree $n - 1$.

When G is isomorphic to K_n, $\kappa'(G) = \delta(G) = n - 1$, we've defined $\kappa(G) = n - 1$ artificially (and this theorem is part of the reason why). So the inequality $\kappa(G) \leq \kappa'(G)$ is satisfied.

Now we just have to carry out our argument assuming G is not a copy of K_n. This means that $\delta(G) < n - 1$, so $\kappa'(G) < n - 1$, so $|\partial(S)| < n - 1$. This means that not all $k(n - k)$ of the possible edges with one endpoint in S are actually present. Therefore we can choose $s \in S$ and $t \notin S$ which are not adjacent, and construct U by the following rule: for each edge $xy \in \partial(S)$, add either x or y to U, taking care not to add either s or t. As we've already shown, this is a vertex cut of size at most $|\partial(S)|$, proving that $\kappa(G) \leq \kappa'(G)$. □

Since this is a section called "Examples", let me reassure you by means of an example that apart from the constraint $\kappa(G) \leq \kappa'(G) \leq \delta(G)$, almost all triples $(\kappa(G), \kappa'(G), \delta(G))$ are possible. There is only one other constraint on such triples: if $\kappa(G) = 0$, it is because G is not connected, so $\kappa'(G) = 0$ as well.

Let $1 \leq a \leq b \leq c$. To find a graph where $\kappa(G) = a$, $\kappa'(G) = b$, and $\delta(G) = c$, begin with two copies of K_{c+1}. Let A be a set of a vertices in the first copy and let B be a set of b vertices in the second copy; add b edges between A and B, making sure to use each vertex in A at least once and each vertex in B exactly once. Call the result $G_{a,b,c,d}$. An example of this construction with $a = 2$, $b = 4$, and $c = 8$ is shown in Figure 26.2.

Question:	Why is $\kappa(G_{a,b,c,d}) = a$?
Answer:	The vertices in A are an a-vertex cut. If we delete fewer than a vertices, we can find vertices $x \in A$ and $y \in B$ that are adjacent, and have not been deleted; since every vertex of $G_{a,b,c,d}$ is adjacent to either x or y, the graph will still be connected.

Question:	Why is $\kappa'(G_{a,b,c,d}) = b$?
Answer:	The edges between A and B are a b-edge cut. If we delete fewer than b edges, then in particular (because $b \leq c$, and $\kappa'(K_{c+1}) = c$) each copy of K_{c+1} is connected; also, at least one edge between the two copies of K_{c+1} remains, connecting them together.

Question:	Why is $\delta(G_{a,b,c,d}) = c$?
Answer:	We started with two copies of K_{c+1}, in which every vertex has degree c. We added edges to $a + b$ vertices, which is at most $2c$; therefore at least two vertices still have degree c.

Question:	Why is there even a d in $G_{a,b,c,d}$?
Answer:	No reason; I'm just messing with you.

26.3 Local connectivity

We often want to solve a problem which is similar to finding a vertex cut or edge cut in a graph, but more specialized.

As I'm writing this chapter, it is November, and in the United States, Thanksgiving is coming up. Many people travel long distances to celebrate Thanksgiving with their family, but it's also almost winter, so if you take a plane across the US, you might have to deal with cancellations due to snow and other winter weather. Some flights might be canceled, deleting an edge in the graph of possible airplane trips. In extreme circumstances, an entire airport might shut down and divert all flights, deleting a vertex.

How many of these events must happen for you to be entirely unable to get to your destination? This is a bit like the question of computing edge connectivity (in the case of flight cancellations) or vertex connectivity (in the case of closed airports).

In practice, vertex and edge connectivity of this graph are low. I can speak to this from personal experience: I have spent three years living in Urbana, Illinois. The local airport has flights to only two other airports: Chicago O'Hare and Dallas–Fort Worth. It was not a very reliable means of travel, especially in the winter!

Suppose, however, that you live in New York City and your family lives in Los Angeles. You do not care if a few cancellations are enough to disconnect you from Urbana, because you're

not going to Urbana. In your case, the number of cancellations that would have to occur for all routes you could take to be blocked off is truly implausible.

To model situations like this, we need a more specialized definition. Here, let s and t be two vertices in a graph G; we will specifically consider what it takes to separate s from t.

- An **$s-t$ vertex cut** (or just **$s-t$ cut**) in G is a vertex cut U that separates s from t: vertices s and t are in different connected components of $G-U$. (In particular, U must not contain either s or t.)

  The **$s-t$ connectivity in G**, denoted $\kappa_G(s,t)$ or just $\kappa(s,t)$ if there is no need to specify the graph, is the smallest number of vertices in an $s-t$ cut.

- An **$s-t$ edge cut** in G is an edge cut X that separates s from t: vertices s and t are in different connected components of $G-X$.

  The **$s-t$ edge connectivity in G**, denoted $\kappa'_G(s,t)$ or just $\kappa(s,t)$, is the smallest number of edges in an $s-t$ edge cut.

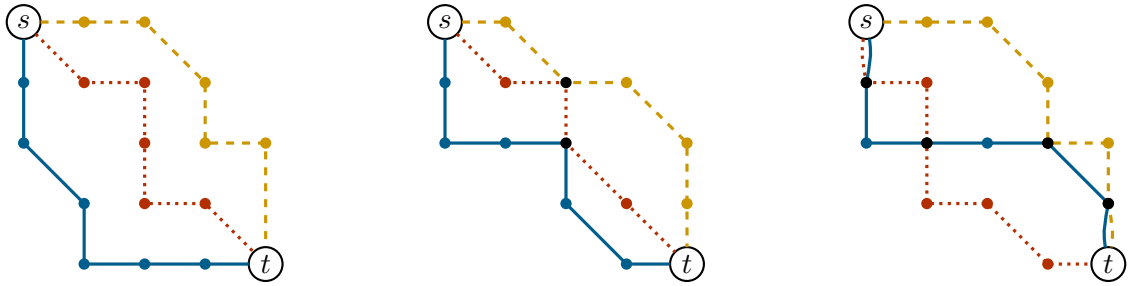
Question:	What happens to $\kappa(s,t)$ if s and t are adjacent?
Answer:	In this case, there is no vertex cut that separates s from t; we either disregard this case entirely or set $\kappa(s,t)=\infty$. However, there are no problems with defining $\kappa'(s,t)$; we just have to make sure that st is one of the edges we delete.

There are, broadly speaking, two reasons to care about $s-t$ cuts of the two types. One reason is an application such as your hypothetical Thanksgiving travel plans from New York City to Los Angeles, where you really only care about whether an $s-t$ path survives. There is another: as we will discover in the next few chapters, finding $\kappa_G(s,t)$ is sometimes much easier than finding $\kappa(G)$. If we really want to know $\kappa(G)$, we might perform several computations of $\kappa_G(s,t)$ instead.

Question:	If you were given a table of $\kappa_G(s,t)$ and $\kappa'_G(s,t)$ for all pairs of vertices s and t, could you use it to determine $\kappa(G)$ and $\kappa'(G)$?
Answer:	Yes $\kappa'(G)$ is the minimum of $\kappa'_G(s,t)$ over all pairs $\{s,t\}$, and if G is not a complete graph, $\kappa(G)$ is the minimum of $\kappa_G(s,t)$ over all non-adjacent pairs $\{s,t\}$.

The quantities $\kappa(s,t)$ and $\kappa'(s,t)$ are the solutions to optimization problems: of all $s-t$ cuts of the appropriate type, we want to find the smallest. Finding upper bounds is, in both cases, straightforward (if not necessarily easy): if you find an $s-t$ cut U, then you know that $\kappa(s,t)\leq|U|$. But how do we prove a lower bound?

Here's one possible answer. Suppose that the graph contains a subgraph such as the one in Figure 26.3a. It is the union of three $s-t$ paths (a blue path, a red path, and a yellow path). Moreover, the three paths share no internal vertices. Then it's clear that $\kappa(s,t)\geq 3$: we need to delete at least one of the internal vertices from each path if we want to disconnect s from t.



(a) Internally disjoint paths (b) Edge-disjoint paths (c) Just some paths

Figure 26.3: Three different sets of three $s - t$ paths

Question:	Why is it important that the three paths share no internal vertices?
Answer:	Consider the paths in Figure 26.3b, instead. They provide three ways to get from s to t, but some vertices are repeated. It's possible to destroy multiple paths by deleting just one vertex, and in fact all three paths can be destroyed by deleting 2 vertices.

The paths in Figure 26.3b are still useful, though: they share no edges. If we seek to disconnect s from t by deleting edges, the existence of such paths proves that at least three edges need to be deleted: one from each path. Compare this to the paths in Figure 26.3c, which have no special properties relative to each other! Here, we can destroy all three paths by deleting just two edges.

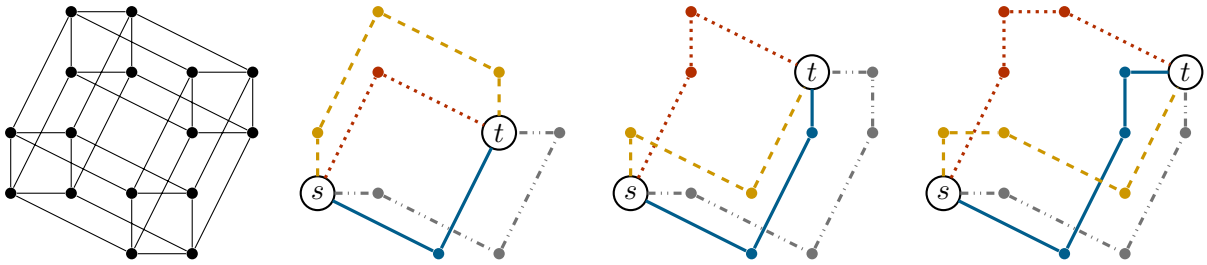
To generalize, we say that two paths in a graph are **internally disjoint** if they do not share any internal vertices; they are **edge-disjoint** if they do not share any edges. I want to emphasize that both of these definitions are properties a pair of paths can have. It would not make sense to look at an $s - t$ path and ask, "Is this $s - t$ path internally disjoint or not?"

For a set of paths, such as what we see in Figure 26.3, the formally correct phrasing would be that the paths in a set are "pairwise internally disjoint" if no two paths share internal vertices, and "pairwise edge-disjoint" if no two paths share edges. The word "pairwise" emphasizes that we consider the paths two at a time, rather than all at once; for a set of 100 paths, it's not enough to know that there's no vertex that all 100 paths pass through.

This is a mouthful. To make it a little bit easier, I will talk about a "set of internally disjoint paths" or a "set of edge-disjoint paths" to mean that any two paths in the set are internally disjoint or edge-disjoint, respectively. In case you are confused, always think back to this example and the argument we are trying to make with the set of paths! We want the paths to be sufficiently separate that to destroy them all (by deleting whichever of vertices or edges we're thinking about) we need to handle each path separately.

We will do a lot more with this idea in the next two chapters. For now, I will use it to go one step further beyond Proposition 26.2.

Proposition 26.4. *The hypercube graph Q_4 is 4-connected.*



(a) Q_4 (b) $\kappa(0000, 0011) \geq 4$ (c) $\kappa(0000, 0111) \geq 4$ (d) $\kappa(0000, 1111) \geq 4$

Figure 26.4: Sets of internally disjoint paths in Q_4

Proof. Our strategy will be to compute $\kappa(Q_4)$ by computing $\kappa(s,t)$ for every s and every t. But first, let me begin with a bit of review of Q_4. This graph has $2^4 = 16$ vertices, which are 4-bit binary strings $b_1b_2b_3b_4$. Two vertices are adjacent if the binary strings agree in 3 out of 4 positions, and only disagree in 1 position.

The hypercube graph Q_4 has many automorphisms. To review some concepts you may not have thought about since Chapter 2, I'll write them out in detail. We'll need two types of automorphisms of Q_4 to reduce the casework we have to do:

- For each position i, let $\varphi_i \colon V(Q_4) \to V(Q_4)$ be the function that flips the i^{th} bit; for example, $\varphi_3(0101) = 0111$. This is a bijection because it is its own inverse. It is an automorphism because when we compare two binary strings x and y, if we flip the i^{th} bit in both of them, it doesn't change which positions they agree in.
- For every two positions i and j, let $\varphi_{ij} \colon V(Q_4) \to V(Q_4)$ be the function that swaps the i^{th} and j^{th} bits: for example, $\varphi_{13}(1001) = 0011$. This is a bijection because it is its own inverse. It is an automorphism because when we compare two binary strings x and y, applying φ_{ij} to both might move a position they disagree in (from i to j or from j to i) but won't change the number of disagreements.

Our goal is to compute $\kappa(s,t)$ for every pair of vertices $\{s,t\}$. There are $\binom{16}{2} = 90$ pairs to test if we do it the hard way. However, by applying the automorphisms above, we can reduce all 90 cases to just a few! The general logic is this: if φ is any automorphism of Q_4, then $\kappa(s,t) = \kappa(\varphi(s), \varphi(t))$.

Question:	How do we know this?
Answer:	If there is a vertex cut U separating s from t, then applying φ to its vertices gives a vertex cut of the same size separating $\varphi(s)$ from $\varphi(t)$. In the other direction, applying φ^{-1} to the vertices of a $\varphi(s) - \varphi(t)$ cut gives an $s - t$ cut of the same size.

Consider any pair $s, t \in V(Q_4)$. First, some composition of the automorphisms $\varphi_1, \ldots, \varphi_4$ maps s to 0000 and t to some vertex t'. Therefore $\kappa(s,t) = \kappa(0000, t')$. The second type of automorphisms (the φ_{ij}'s) leave 0000 unchanged, but some composition of them sorts the bits of t' in ascending order: to one of 0001, 0011, 0111, or 1111. Therefore

$$\kappa(Q_4) = \min\Big\{\kappa(0000, 0001),\ \kappa(0000, 0011),\ \kappa(0000, 0111),\ \kappa(0000, 1111)\Big\}.$$

Of these four cases, $\kappa(0000, 0001) = \infty$ because 0000 and 0001 are adjacent; we can skip this one. For the other three, the vertex cut $\{0001, 0010, 0100, 1000\}$ consisting of all four neighbors of 0000 separates 0000 from the other vertices, so they are all at most 4. Finally, Figure 26.4 shows that in all three of these cases, we can find a set of 4 internally disjoint paths. Therefore 4 is a lower bound as well, and we conclude that $\kappa(Q_4) = 4$. □

Question:	What is $\kappa'(Q_4)$?
Answer:	By Theorem 26.3, it is sandwiched between $\kappa(Q_4)$ and $\delta(Q_4)$. Both of these are 4, so $\kappa'(Q_4) = 4$ as well.

26.4 Duality

The technique of internally disjoint paths seems pretty powerful based on this example, but there is an unresolved question. If we use this technique to prove lower bounds on $\kappa(s, t)$ or $\kappa'(s, t)$, will we always be able to prove a lower bound equal to the true value of these numbers? Or is it possible that in some cases, $\kappa(s, t) = 4$, but the largest set of internally disjoint paths only contains 3 paths?

In the next chapter, we will prove that this will never happen: not for vertices, and not for edges. In preparation, let me tell you about one framework to think about the relationship between $s - t$ cuts and sets of disjoint paths: optimization duality.

Optimization duality is a relationship that can exist between two optimization problems: a maximization problem and a minimization problem. We want to be fairly general here, but not so general that we get lost in abstraction. Let's say that:

- The maximization problem is, given a set $\mathcal{X}$ and a function $f\colon \mathcal{X} \to \mathbb{R}$, to find an element $x \in \mathcal{X}$ such that $f(x)$ is as large as possible. Written concisely, it is the problem of finding $\max\{f(x) : x \in \mathcal{X}\}$.
- The minimization problem is, given a set $\mathcal{Y}$ and a function $g\colon \mathcal{Y} \to \mathbb{R}$, to find an element $y \in \mathcal{Y}$ such that $g(y)$ is as small as possible. Written concisely, it is the problem of finding $\min\{g(y) : y \in \mathcal{Y}\}$.

Let's try putting some problems we've already studied into this language. Don't worry too much about how the functions f and g are implemented; it's enough to know they exist.

Question:	In the maximization problem of finding the matching number of a graph G, what is $\mathcal{X}$ and what is f?
Answer:	The set $\mathcal{X}$ is the set of all matchings in G. If $x \in \mathcal{X}$ is such a matching, then $f(x)$ is the number of edges in it.

Question:	In the minimization problem of finding the chromatic number of a graph G, what is $\mathcal{Y}$ and what is g?
Answer:	The set $\mathcal{Y}$ is the set of all proper colorings of G. If $y \in \mathcal{Y}$ is such a coloring, then $g(y)$ is the number of colors used by y.

Optimization duality comes in two forms: weak and strong. To begin with, we say that a maximization problem $\max\{f(x) : x \in \mathcal{X}\}$ and a minimization problem $\min\{g(y) : y \in \mathcal{Y}\}$ are in *weak duality* if $f(x) \geq g(y)$ for every $x \in \mathcal{X}$ and every $y \in \mathcal{Y}$. Equivalently,

$$\max\{f(x) : x \in \mathcal{X}\} \leq \min\{g(y) : y \in \mathcal{Y}\}$$

is the inequality that characterizes weak duality.

Why is this helpful? Well, consider the example of matchings and vertex covers in graphs: in the entire textbook, this is perhaps our best example of dual optimization problems. By Proposition 13.4, whenever M is a matching and U is a vertex cover in the same graph G, we have $|E(M)| \leq |U|$: the maximization problem is always bounded above by the minimization problem. This means we can try to find small vertex covers to get upper bounds on the size of a matching: upper bounds that are otherwise hard to come by.

In general, duality fills a gap in our understanding of an optimization problem. By default, if we have an optimization problem to maximize $f(x)$ over all $x \in X$, and we've worked on it for a while, we might not have anything to show for our progress except some element $x^* \in X$ for which $f(x^*)$ is the largest found so far. This proves a lower bound on the optimal value: $\max\{f(x) : x \in \mathcal{X}\}$ is at least $f(x^*)$.

To prove an upper bound on $\max\{f(x) : x \in \mathcal{X}\}$, we could try every element of $\mathcal{X}$ to see which is best. Or, we could come up with and prove an insightful theorem that gives a general upper bound. These both seem very hard. But if we have a dual problem $\min\{g(y) : y \in \mathcal{Y}\}$, then we can switch to working on that problem, instead! After a while, the element $y^* \in Y$ for which $g(y^*)$ is the smallest found so far is probably pretty good. This value $g(y^*)$ will be an upper bound on our original problem: by weak duality, $\max\{f(x) : x \in \mathcal{X}\}$ is at most $g(y^*)$.

The reason weak duality is "weak" is that the bounds we get this way might turn out to be very bad. Even if we've found the best x^* we can, weak duality does not guarantee us a way to ever prove that it's the best. If the two dual problems meet in the middle, and

$$\max\{f(x) : x \in \mathcal{X}\} \leq \min\{g(y) : y \in \mathcal{Y}\},$$

we say that they are in *strong duality*.

In the worst of all possible worlds, there is no structure to $\mathcal{X}$, no meaning to f, and no way to find $\max\{f(x) : x \in \mathcal{X}\}$ except by exhaustive search. In such a world, even if you find the optimal solution x^* after hundreds of hours of computing time, skeptics won't trust you: they will say that maybe your software had a bug in it, and you missed some cases. But with strong duality,[36] you can find an optimal solution y^* to the dual problem. Now you can silence the skeptics by simply checking that $f(x^*) = g(y^*)$; this is a (hopefully) quick way of demonstrating that both solutions are optimal.

Again, matchings and vertex covers are an ideal example, because the duality here is sometimes, but not always strong.

Question:	When is there strong duality between finding the largest matching in G and finding the smallest vertex cover in G?
Answer:	By Kőnig's theorem (Theorem 14.2), strong duality holds when G is bipartite.

[36] And possibly hundreds of hours more of computing time.

(By the way, you should brush up on Kőnig's theorem in preparation for the next chapter; it will come in handy!)

So how does this apply to vertex and edge connectivity? The version of the problem that exhibits duality is the problem with cuts and paths between two specific vertices s and t; the "global" version is not as nice.

Start with the problem of finding $\kappa_G(s,t)$. This is a minimization problem: we are looking for the smallest $s-t$ cut in G. The corresponding maximization problem is the problem of finding the largest set of internally disjoint $s-t$ paths. So far, we know:

Proposition 26.5. *Let s and t be two vertices in a graph G. If there is a set of k internally disjoint $s-t$ paths in G, then $\kappa_G(s,t) \geq k$: there is no $s-t$ cut with fewer than k vertices.*

Proof. Take any set U of fewer than k vertices, with $s,t \notin U$, that we suspect might be an $s-t$ cut. Every vertex $x \in U$ lies on at most one of the $s-t$ paths in our set, because they are internally disjoint. There are k paths, and fewer than k vertices in U, so there is at least one path with no vertices of U on it.

That path still exists in $G-U$, proving that s and t are in the same connected component of $G-U$. So U is not an $s-t$ cut, after all. □

Question:	What kind of duality does Proposition 26.5 give us?
Answer:	Weak duality! The proposition gives an inequality between the largest number of internally disjoint $s-t$ paths and the smallest number of vertices in an $s-t$ cut, but it does not guarantee that they meet in the middle. So far, we know of no reason why there can't be a graph G with only 10 internally disjoint $s-t$ paths, but no $s-t$ cut with fewer than 20 vertices.

Similarly, we can prove weak duality between the minimization problem of finding the smallest $s-t$ edge cut, and the maximization problem of finding the largest set of edge-disjoint $s-t$ paths:

Proposition 26.6. *Let s and t be two vertices in a graph G. If there is a set of k edge-disjoint $s-t$ paths in G, then $\kappa'_G(s,t) \geq k$: there is no $s-t$ edge cut with fewer than k edges.*

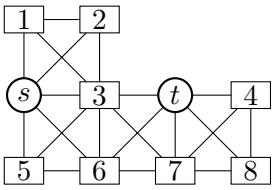
Proof. As before, but replacing vertices by edges. The key point is that every edge in G lies on at most one path in our set. A set of fewer than k edges cannot be an $s-t$ edge cut, because one of our $s-t$ paths will avoid it. □

Question:	Is there also weak duality between $s-t$ edge cuts and internally disjoint $s-t$ paths?
Answer:	Yes! Internally disjoint $s-t$ paths are, in particular, edge-disjoint. But whether or not strong duality will hold, we'd like to aim for the "least weak" weak duality, so we don't want to restrict our maximization problem unnecessarily. The most general sets of $s-t$ paths that will work to give a lower bound on $\kappa'_G(s,t)$ are the sets of edge-disjoint paths, so we want to use those.

In Chapter 27, we will prove that in both cases, strong duality holds: a result known as Menger's theorem.

26.5 Practice problems

1. Let G be the graph shown below.



   a) Find the vertex connectivity $\kappa(G)$.

   b) Find the $s-t$ edge connectivity $\kappa'_G(s,t)$.

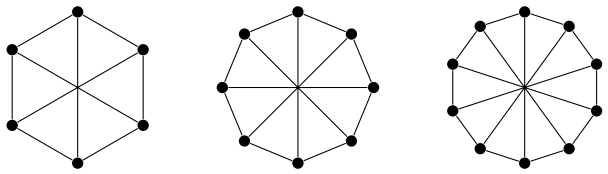
2. a) Prove that $\kappa(Q_4) = 4$ directly: by showing that no set of 3 deleted vertices can disconnect Q_4. Using Proposition 26.2 may help.

   b) Prove that $\kappa(Q_n) = n$ by induction on n, either directly or using the idea of internally disjoint paths.

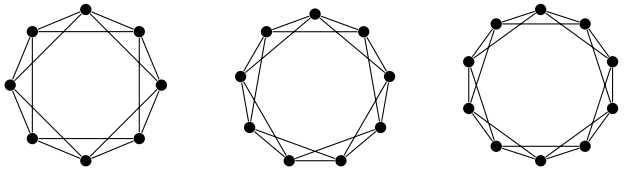
3. In Chapter 5, the *Harary graph* $H_{n,r}$ was a particular circulant graph we used in Theorem 5.1 to give an example of an r-regular graph on n vertices.

   In fact, more is true: whenever $0 \leq r \leq n-1$ and at least one of r and n is even, the Harary graph $H_{n,r}$ provides an example of an r-regular graph with $\kappa(H_{n,r}) = r$. It achieves the minimum possible number of edges in an r-connected n-vertex graph.

   a) Prove that $\kappa(H_{n,r}) = r$ for all even n when $r = 3$. The Harary graphs $H_{6,3}$, $H_{8,3}$, and $H_{10,3}$ are shown as examples below:



b) Prove that $\kappa(H_{n,r}) = r$ for all n when $r = 4$. The Harary graphs $H_{8,4}$, $H_{9,4}$, and $H_{10,4}$ are shown as examples below:



c) If you are feeling confident, prove that $\kappa(H_{n,r}) = r$ for all valid choices of n and r.

d) Give an example of a connected r-regular circulant graph which is not r-connected. (The smallest example has 8 vertices.)

4. The Petersen graph is 3-connected.

   a) Prove this by picking two vertices of the Petersen graph that are not adjacent, and finding a set of three internally disjoint paths between them.

   b) Prove this by picking a vertex of the Petersen graph, deleting it, and finding an ear decomposition of the remaining graph.

   c) Explain, using automorphisms of the Petersen graph, why either of the strategies above is a fully general proof that the Petersen graph is 3-connected.

5. A graph is called fragile if it has a vertex cut which is an independent set.

   a) Prove that the clique number of a graph is 2, then it is fragile.

   b) For all $n \geq 3$, find an example of a graph with n vertices and $2n - 3$ edges which is not fragile.

   (This exercise is based on some observations in "A note on fragile graphs" by Guantao Chen and Xingxing Yu [15].)

6. Let D be a *strongly connected digraph* with at least 3 vertices (as defined in Chapter 12), and let G be its *underlying graph* (as defined in Chapter 7). You may either assume that D never contains both arcs (x, y) and (y, x), or make G a multigraph with two copies of edge xy in this case.

   a) Prove that $\kappa'(G) \geq 2$.

   b) Give an example showing that it is not necessarily true that $\kappa(G) \geq 2$.

7. Without making use of symmetry, the strategy used to prove Proposition 26.4 would seem hopeless, because there are too many pairs (s, t) for which $\kappa(s, t)$ needs to be verified. However, it is often enough to check a smaller, but well-chosen set of pairs. The following strategy is due to Abdol Esfahanian and S. L. Hakimi [31].

   Let x be an arbitrary vertex of a graph G which is not complete, and suppose that the following is true:

   - For every vertex y not adjacent to x, $\kappa_G(x, y) \geq k$.
   - For every two vertices y, z that are adjacent to x but not to each other, $\kappa_G(y, z) \geq k$.

   Prove that $\kappa(G) \geq k$.

8. It is tempting to guess that the edge connectivity $\kappa'(G)$ is equal to $\kappa(L(G))$: the vertex connectivity of the line graph of G. However, this is false.

   a) Prove that every vertex cut in $L(G)$ corresponds to an edge cut in G. Deduce that $\kappa'(G) \leq \kappa(L(G))$.

   b) Not every edge cut in G is a vertex cut in $L(G)$. Think about how this statement might fail, and use it to find an example where $\kappa'(G) < \kappa(L(G))$. (A relatively small example exists with 5 vertices and 8 edges.)

   c) Is there a way to compute $\kappa'(G)$ using only $\kappa(L(G))$ and some simple information about G?

27 Menger's theorem

The purpose of this chapter

Menger's theorem answers the questions posed at the end of the previous chapter, and it is the main reason why $s-t$ connectivity ($\kappa_G(s,t)$ and $\kappa'_G(s,t)$ is studied): even if you want to know how hard it is to disconnect s from t for practical purposes, Menger's theorem goes a long way toward assuring you that $s-t$ connectivity is a useful model. Beyond that, it helps us understand $\kappa(G)$ (and $\kappa'(G)$) as well, often through Dirac's fan lemma.

In an introductory course on graph theory, it might be useful to have a plan for covering just one portion of this chapter, rather than the whole thing. I can think of two good ways to do this. One option is to make the proof of Menger's theorem the final goal of the semester. It is suitably dramatic, since it is a result that generalizes previous ones about matchings, answers questions posed in the previous chapter, and is one of the most difficult proofs in this book, if covered in detail.

Another option is to avoid proving Menger's theorem, and instead focus on applications of it. I've chosen to include Theorem 27.9 (the Chvátal–Erdős theorem) as an example of this, for several reasons. It connects the vertex connectivity $\kappa(G)$ to two important topics from earlier chapters: Hamilton cycles and independent sets. Also, seeing the technique in the proof (illustrated in Figure 27.4) is useful: it shows up over and over again in the study of Hamilton cycles. If taking this option, it would still be good to prove Lemma 27.2 if there's time: the connection between Menger's theorem and matchings is also an important application of Menger's theorem!

Keep in mind that this chapter relies heavily on many different concepts from previous parts of the book: Kőnig's theorem from Chapter 14, Hamilton cycles from Chapter 17, independent sets from Chapter 18, and line graphs from Chapter 20.

27.1 Menger's theorem

We ended the previous chapter with a discussion of optimization duality, and two instances of it in particular:

- The duality between minimizing the number of vertices in an $s-t$ cut, and maximizing the size of a set of internally disjoint $s-t$ paths.
- The duality between minimizing the number of edges in an $s-t$ edge cut, and maximizing the size of a set of edge-disjoint $s-t$ paths.

We proved in Proposition 26.5 and Proposition 26.6 that at the very least, *weak duality* holds in both pairs: the largest set of paths is a lower bound on the number of vertices or edges in the cut.

In 1927, Karl Menger proved [72] that *strong duality* holds in both pairs: the largest set of paths is equal to the number of vertices or edges in the cut.[37] We will begin with the vertex version of Menger's theorem. I will state it in the following form, letting Proposition 26.5 (which we've already proved) provide the converse:

Theorem 27.1 (Menger's theorem)**.** *If s and t are any two non-adjacent vertices of a graph G, then G contains a set of $\kappa_G(s,t)$ internally disjoint $s-t$ paths.*

The main goal of this chapter is to prove Menger's theorem.

Many proofs of Menger's theorem are known, so let me summarize the origin of the proof presented here. The argument by minimal counterexample in Lemmas 27.3, 27.4, and 27.5 is due to Dirac [23]. The remaining cases can be solved using Kőnig's theorem (Theorem 14.2), which is also how the proof is finished in West's *Introduction to Graph Theory* [104].

I prefer to do things this way, because Lemma 27.2 can be used to go in either direction: it can also let us deduce Kőnig's theorem from Menger's theorem. However, we do not really need the full power of Kőnig's theorem; Hall's theorem would be enough, and in fact Dirac finished the proof without using any additional results.

The relationship between Menger's theorem and our earlier results about matchings extends to algorithms. If we have a complete algorithm for computing $\kappa_G(s,t)$, which finds both an $s-t$ cut of $\kappa_G(s,t)$ vertices and a set of $\kappa_G(s,t)$ internally disjoint $s-t$ paths, then we can turn it into a matching algorithm. We do not yet have such an algorithm, and the proof of Menger's theorem in this chapter is not suitable for finding one. However, in Chapter 28, we will look at the maximum flow problem, which can be used to solve both problems. The algorithm this gives is more or less the one in Chapter 14, but it's now a special case of a more general method.

27.2 Kőnig-type graphs

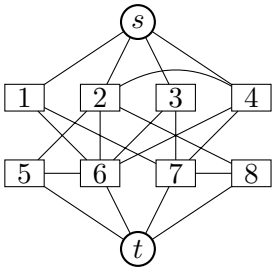
We will begin by understanding $s-t$ cuts in the special case of what I will temporarily call "Kőnig-type" graphs. This is not a standard term, just my name for the case of Menger's theorem we will address using Kőnig's theorem.

We will say that a graph G with two vertices designated s and t is *Kőnig-type* if the following is true:

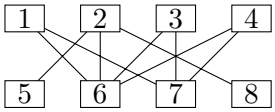
1. Vertices s and t are not adjacent. (This is a prerequisite for $\kappa_G(s,t)$ to be an interesting problem in the first place: if s and t are adjacent, then $\kappa_G(s,t)=\infty$.)
2. Every other vertex in the graph is adjacent to s or to t, but not both.

An example of a Kőnig-type graph is given in Figure 27.1a. Vertices $1,2,3,4$ are adjacent to s; vertices $5,6,7,8$ are adjacent to t.

[37] In the vertex version, we must assume that the vertices s and t are not adjacent, or else there is no $s-t$ cut at all. This will be a recurring assumption for a large part of this chapter.



(a) A Kőnig-type graph



(b) The corresponding bipartite graph

Figure 27.1: The correspondence between Kőnig-type graphs and bipartite graphs

Question:	In Figure 27.1a, the sets $\{1, 2, 3, 4\}$ and $\{5, 6, 7, 8\}$ are both $s - t$ cuts. Is there a smaller $s - t$ cut?
Answer:	Yes: the 3-vertex set $\{2, 6, 7\}$ is the unique smallest $s - t$ cut.

Question:	Is there a set of 3 internally disjoint $s - t$ paths to match it?
Answer:	Yes, represented by the walks $(s, 2, 5, t)$, $(s, 3, 6, t)$, and $(s, 4, 7, t)$.

Given a Kőnig-type graph G, let A be the set of vertices adjacent to s, and let B be the set of vertices adjacent to t. We construct a bipartite graph H from G by deleting vertices s and t, as well as all edges not between A and B. Figure 27.1b shows the result of applying this to the Kőnig-type graph in Figure 27.1a. (The deletion of vertices s and t is hard to miss, but edges 24, 56, and 78 have also been deleted, which is more subtle.)

Kőnig's theorem says that $\alpha'(H)$, the largest number of edges in a matching in H, is equal to $\beta(H)$, the smallest number of vertices in a vertex cover of H.

Question:	In Figure 27.1b, can you find a matching and a vertex cover of equal size?
Answer:	Edges $\{16, 25, 37\}$ form a matching, and vertices $\{2, 6, 7\}$ are a vertex cover of the same size: every edge is incident to at least one of these three vertices.

The vertex cover we found is the same set as the $s - t$ cut in Figure 27.1a. This is not a coincidence! Since $\{2, 6, 7\}$ is a vertex cover of H, deleting these three vertices from G also eliminates all edges between A and B, which leaves no way to get from s to t.

More precisely and more generally, let's state an equivalence:

Lemma 27.2. *Menger's theorem for a Kőnig-type graph G is equivalent to Kőnig's theorem for the associated bipartite graph H.*

Proof. We've already seen a glimpse of the relationship between $s-t$ cuts in G and vertex covers in H, but it will be easier to prove it formally if we first understand the relationship between internally disjoint $s-t$ paths in G and matchings in H.

A matching M in H, like the matching with $E(M) = \{16, 25, 37\}$ we found in Figure 27.1b, can always be turned into a set of internally disjoint $s-t$ paths of the same size. For each edge $xy \in E(M)$, take the path represented by (s, x, y, t). The internal vertices of this path are precisely the endpoints of xy, and by the definition of a matching, two edges in M share no endpoints. This means that the paths we get in this way are always internally disjoint.

Not all $s-t$ paths are like this; in Figure 27.1a, there are very long paths like the one represented by $(s, 2, 4, 7, 3, 6, 5, t)$. However, each $s-t$ path must contain an edge xy, where $x \in A$ and $y \in B$, or else it will never reach t. Given a set of internally disjoint $s-t$ paths, let M be the subgraph of H containing the first edge between A and B from each path. Because the paths were internally disjoint, these edges share no endpoints, so M is a matching.

This correspondence shows that the largest number of internally disjoint $s-t$ paths in G is equal to the matching number $\alpha'(H)$. For $s-t$ cuts in G and vertex covers in H, the relationship appears to be much more simple: they are one and the same! But why is this?

In both cases, we are looking for a set of vertices not including s or t; a subset $U \subseteq V(H)$. The definition of a vertex cover is simpler to check than the definition of an $s-t$ cut: U is a vertex cover in H if and only if it contains an endpoint of every edge xy with $x \in A$ and $y \in B$. So let's check that this also describes when U is an $s-t$ cut in G.

Question:	If U is an $s-t$ cut in G, why must it contain at least one endpoint of every edge xy with $x \in A$ and $y \in B$?
Answer:	If not, then the walk (s, x, y, t) shows that s and t are still in the same connected component of $G - U$.

Question:	If U contains at least one endpoint of every such edge, why is it an $s-t$ cut in G?
Answer:	In this case, $G - U$ has no edges between A and B (or what's left of them) remaining, so there is no way to get from s to t.

This proves that $s-t$ cuts in G are exactly the same as vertex covers in H; in particular, $\kappa'_G(s,t) = \beta(H)$.

We're now ready to prove the equivalence claimed in the lemma. We have two pairs of equal quantities, one in G and one in H:

1. The largest size of a set of internally disjoint $s-t$ paths in G is equal to the matching number $\alpha'(H)$.
2. The $s-t$ connectivity $\kappa_G(s,t)$ is equal to the vertex cover number $\beta(H)$.

Both Menger's theorem for G and Kőnig's theorem for H say that the first pair is also equal to the second pair, so the claims made by the two theorems are equivalent. □

27.3 Reducing all other cases

To obtain a proof of Menger's theorem in general, we must now deal with graphs which are not Kőnig-type.

We will use a proof by minimal counterexample: a sort of combination of a proof by induction and a proof by contradiction. Suppose for sake of contradiction that Menger's theorem is false for some graphs. Then we may take G to be a counterexample to Menger's theorem which has as few vertices as possible: a minimal counterexample.

Sometimes a minimal counterexample is also called a "minimal criminal", but I don't like this terminology, because it's too much of a tongue-twister when I try to say it out loud while teaching class. I don't mind writing it in a book, though.

What do we do with a minimal counterexample? We try to deduce additional properties: for example, right now, in the case of Menger's theorem, we know that a minimal counterexample is not Kőnig-type, because then it wouldn't be a counterexample. The reason minimality is so important is that very often, we can say, "A minimal counterexample must have property X, because otherwise, we could modify it in Y way and obtain a smaller counterexample."

Proofs by minimal counterexample can often be turned into proofs by induction, but I want to include a proof of this type here, because they're a very important exploration technique. When you don't yet know whether a theorem is true, proving some properties of a minimal counterexample is useful in two ways: it is partial progress toward a proof of the theorem (if it's true), but it can also help us find a counterexample (if it's false).

So let's see how it works here. The following three lemmas all have the common hypothesis that a graph G, with designated non-adjacent vertices s and t, is a minimal counterexample to Menger's theorem. (I will not keep restating these assumptions about G each time.) We will prove a few properties of G, and then put them together to prove Menger's theorem. I should warn you that (as in every book in which the hero has to accomplish three tasks) the third lemma is the trickiest.

Lemma 27.3. *G cannot contain a vertex x adjacent to both s and t.*

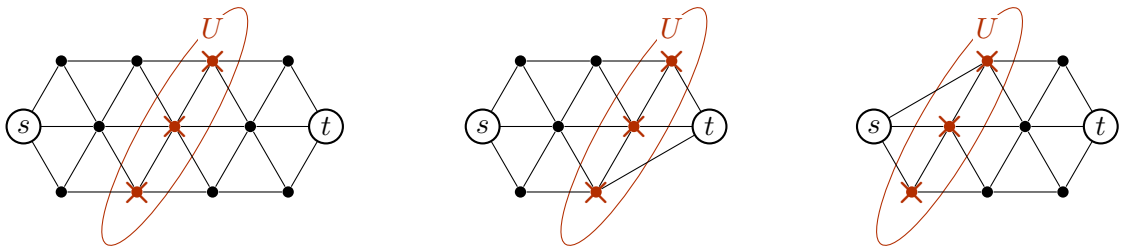
Proof. Suppose for the sake of contradiction that G has such a vertex. Then x must be part of every $s-t$ cut: if it is not deleted, there is an $s-t$ path P represented by (s,x,t). So every $s-t$ cut has the form $U \cup \{x\}$, where U is an $s-t$ cut in $G-x$.

Let $H = G - x$. Because (as we've just proved) every $s-t$ cut in G consists of an $s-t$ cut in H, plus one extra vertex, we have $\kappa_G(s,t) = \kappa_H(s,t) + 1$.

But we know that H is smaller than the minimal counterexample to Menger's theorem! So by Menger's theorem, H contains a set of $\kappa_H(s,t)$ internally disjoint $s-t$ paths.

All these $s-t$ paths are also $s-t$ paths in G, and we can add one more path to the set: the path P defined above. Therefore G contains a set of $\kappa_H(s,t)+1$ internally disjoint $s-t$ paths. But this is exactly the number of paths Menger's theorem promises, so we contradict our assumption that G is a counterexample to Menger's theorem. □

Lemma 27.4. *Every vertex in G, other than s and t, is part of some $s-t$ cut of size $\kappa_G(s,t)$.*



(a) The original graph G (b) Subproblem 1 (L) (c) Subproblem 2 (R)

Figure 27.2: The subproblems in Lemma 27.5

Proof. Let x be an arbitrary vertex of G, and let $H = G - x$. As in the previous lemma, H is smaller than the minimal counterexample to Menger's theorem, so H contains a set of $\kappa_H(s,t)$ internally disjoint $s-t$ paths.

Therefore G also contains a set of $\kappa_H(s,t)$ internally disjoint $s-t$ paths. Since G is a counterexample to Menger's theorem, these paths must not be enough: $\kappa_G(s,t)$ must be bigger than $\kappa_H(s,t)$. In other words, $\kappa_G(s,t) \geq \kappa_H(s,t) + 1$.

Let U be an $s-t$ cut in H of size $\kappa_H(s,t)$. Then $U \cup \{x\}$ is an $s-t$ cut in G of size $\kappa_H(s,t)+1$, because $G-(U\cup\{x\})$ is exactly the same graph as $H-U$. This proves that $\kappa_G(s,t) \leq \kappa_H(s,t)+1$, and since we have the reverse inequality already, we know that $\kappa_G(s,t)$ and $\kappa_H(s,t)+1$ are equal.

Therefore $U \cup \{x\}$ is the $s-t$ cut containing x we wanted: it has size $\kappa_H(s,t)+1 = \kappa_G(s,t)$. □

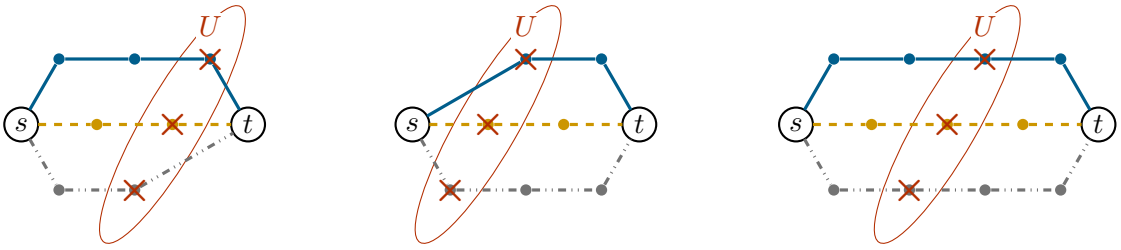
Let A be the set of all vertices adjacent to s in G, and let B be the set of all vertices adjacent to t. From Lemma 27.3, we know that A and B are disjoint, but we do not (yet) know if they include every vertex other than s and t. Both A and B are $s-t$ cuts, though they might have more than $\kappa_G(s,t)$ vertices.

Lemma 27.5. *G has no $s-t$ cuts of size $\kappa_G(s,t)$ not equal to either A or B.*

Proof. Let U be an $s-t$ cut in G with $|U| = \kappa_G(s,t)$. If U contains all of A, then U must be equal to A, because otherwise A would be a smaller $s-t$ cut. Similarly, if U contains all of B, then U must be equal to B. So assume, for the sake of contradiction, that U does not contain all of either set: there is some $x \in A$ and some $y \in B$ such that U does not contain either x or y.

The vertices x and y will be important eventually, but for the moment, we simply use U to split up the problem of finding internally disjoint $s-t$ paths in G into two smaller subproblems. This is shown by example in Figure 27.2, which is worth a thousand words. Here are a few words to describe what we do in general:

Subproblem 1. Take the subgraph of G induced by U and all the vertices in the same connected component of $G-U$ as s. This does not include t, because in $G-U$, vertices s and t are in different components. But now, add t back in, with edges from every vertex in U directly to t. Call the resulting graph L (for "left", since in our example, it retains the left half of Figure 27.3).



(a) $s-t$ paths in L (b) $s-t$ paths in R (c) $s-t$ paths in G

Figure 27.3: Recombining the subproblems in Lemma 27.5

Subproblem 2. Take the subgraph of G induced by U and all the vertices in a different connected component of $G-U$ from s. (This includes t, but not s.) Add s back in, with edges from s directly to every vertex in U. Call the resulting graph R (for "right").

The vertices x (in A, but not in U) and y (in B, but not in U) guarantee that the two subproblems are strictly smaller than G: in L, there is no vertex y, and in R, there is no vertex x. As in Lemma 27.3, the natural next step is to apply Menger's theorem to L and R. To do this, the first thing we need to know is $\kappa_L(s,t)$ and $\kappa_R(s,t)$.

Question:	In Figure 27.2, how do $\kappa_L(s,t)$ and $\kappa_R(s,t)$ compare to $\kappa_G(s,t)$?
Answer:	All three $s-t$ connectivities are equal.

This is true in general. To find an $s-t$ cut in L, we must block all ways to get from s to U (since every vertex to U is adjacent to t). This also blocks all ways to get from s to U in G (since before we pass through U, we cannot reach a place where G and L disagree), so we get an $s-t$ cut in G as well, proving $\kappa_L(s,t) \geq \kappa_G(s,t)$. Similarly, $\kappa_R(s,t) \geq \kappa_G(s,t)$, because an $s-t$ cut in R blocks all ways to get from U to t, which makes it an $s-t$ cut in G as well.

The reverse inequalities $\kappa_L(s,t) \leq \kappa_G(s,t)$ and $\kappa_R(s,t) \leq \kappa_G(s,t)$ hold because U is a cut of size $\kappa_G(s,t)$ in all three graphs: G, L, and R.

Because L is strictly smaller than G, Menger's theorem holds for L, giving a set of $\kappa_G(s,t)$ internally disjoint $s-t$ paths in L. Since $|U| = \kappa_G(s,t)$ and every $s-t$ path in L must first pass through U, this set of paths uses each vertex in U exactly once. By removing the last step from each path, we get a set of internally disjoint paths from s to every vertex of U.

Because R is strictly smaller than G, Menger's theorem holds for R, giving a set of $\kappa_G(s,t)$ internally disjoint $s-t$ paths in R. Since $|U| = \kappa_G(s,t)$ and every $s-t$ path in R must first pass through U, this set of paths uses each vertex in U exactly once. By removing the first step from each path, we get a set of internally disjoint paths from every vertex of U to t.

For each vertex $u \in U$, the union of an $s-u$ path from the first set and a $u-t$ path from the second set is an $s-t$ path in G. (This is shown for our example in Figure 27.3.) All $\kappa_G(s,t)$ paths obtained in this way are internally disjoint: they cannot intersect in or before U, because that would be an intersection in L, and they cannot intersect after U, because that would be an intersection in R.

But finding a set of $\kappa_G(s,t)$ internally disjoint $s-t$ paths in G contradicts our choice of G: that it was a minimal counterexample to Menger's theorem. Therefore the $s-t$ cut U, which we assumed to exist at the beginning of this proof, cannot exist in G. □

Take a deep breath: Lemma 27.3 was quick to prove, and Lemma 27.4 was not bad either, but Lemma 27.5 might have been more than we bargained for. But now, we are ready to finish the proof of Menger's theorem in just a few more steps!

Proof of Menger's theorem (Theorem 27.1). Suppose for the sake of contradiction that a minimal counterexample to Menger's theorem exists: a graph G with designated not-adjacent vertices s and t which does not have a set of $\kappa_G(s,t)$ internally disjoint $s-t$ paths.

By Lemma 27.2 and Kőnig's theorem, we know that G (a counterexample to Menger's theorem) cannot be a Kőnig-type graph.

Question:	Looking back at the definition of Kőnig-type graphs earlier in this chapter, what does it mean that G is not one?
Answer:	The condition for G to be Kőnig-type is that every vertex other than s or t is adjacent to s or t, but not both. So if G is not Kőnig-type, it must have a vertex (other than s or t) which is either adjacent to both s and t, or to neither.

Let x be one such vertex. If x is adjacent to both s and t, this contradicts Lemma 27.3. If x is adjacent to neither s nor t, then we need to work harder. First, apply Lemma 27.4 to find a vertex cut U of size $\kappa_G(s,t)$ with $x \in U$.

Question:	Why does this contradict Lemma 27.5?
Answer:	Lemma 27.5 says that in this case, $U = A$ (the set of neighbors of s) or $U = B$ (the set of neighbors of t). But neither one is possible, because $x \in U$ and x is not adjacent to s or t.

In all case, G contradicts one of the lemmas we proved, so we conclude that a counterexample to Menger's theorem does not exist. □

27.4 Extensions

Many variants of Menger's theorem exist. To begin with, a version of the theorem for $s-t$ edge cuts also exists, which is the converse of Proposition 26.6.

Theorem 27.6 (Menger's theorem, edge version)**.** *If s and t are any two vertices of a graph G, then G contains a set of $\kappa'_G(s,t)$ edge-disjoint $s-t$ paths.*

It is very tempting (and almost works) to try to prove this version of Menger's theorem by applying Theorem 27.1 to the line graph $L(G)$.

Question:	What is the main problem with trying to relate $\kappa'_G(s,t)$ to $\kappa_{L(G)}(s,t)$?
Answer:	The line graph $L(G)$ doesn't have vertices called s and t: its vertices are edges of the original graph!

To fix the problem, we must modify $L(G)$ a little.

Proof of Theorem 27.6. Define a new graph H in one of the following two equivalent ways:

- Construct G' from G by adding two new vertices: one adjacent only to s and one adjacent only to t. Then, let $H = L(G)$.
- Construct H from $L(G)$ by adding two new vertices s^* to t^*. For every vertex of $L(G)$ which represents an edge incident to s, make it adjacent to s^* in H; for every vertex of $L(G)$ which represents an edge incident to T, make it adjacent to t^* in H.

Question:	If we construct H in the first way, what are the two vertices of H that correspond to s^* and t^*?
Answer:	They come from the edges of G' incident to the two newly-added vertices.

We must resolve an issue that will otherwise repeatedly bother us throughout this proof: the relationship between $s^* - t^*$ paths in H and $s - t$ paths in G. In one direction, things work out nicely. Given a walk $(x_0, x_1, \ldots, x_l)$ in G where $x_0 = s$ and $x_1 = t$, the sequence $(s^*, x_0x_1, x_1x_2, \ldots, x_{\ell-1}x_\ell, t^*)$ is an $s^* - t^*$ walk in H, and if the first walk represents a path, so does the second. (Check this!)

Question:	What about the other direction; is it true that every $s^* - t^*$ path in H can be turned into an $s - t$ path in G by reversing this process?
Answer:	No! If it's been a while since you thought about paths in $L(G)$, you should go back and re-read Chapter 20, but walks in a line graph can do some things that walks in the original graph cannot: they can contain segments of the form $(\ldots, xy, xz, xw, \ldots)$ with several edges all incident to the same vertex of G.

However, given an $s^* - t^*$ path in H, we can look at the subgraph of G containing all the edges that were internal vertices of the $s^* - t^*$ path, and all their endpoints. This is a connected subgraph of G containing s and t, so within it we can find an $s - t$ path.

By applying Menger's theorem to H, we get a set of $\kappa_H(s^*, t^*)$ internally disjoint $s^* - t^*$ paths. We know how to turn each of them into an $s - t$ path in G, but we want to know that we'll get a set of edge-disjoint paths when we do so.

Question:	Suppose P and Q are two internally disjoint $s^* - t^*$ paths in H. When we use them to find two $s - t$ paths in G, why are those paths edge-disjoint?
Answer:	Our first steps with P and Q are to go to connected subgraphs of G whose edges correspond to internal vertices of P and of Q. So even those connected subgraphs have no edges in common! Therefore the paths we find in them are also edge-disjoint.

We've found a set of $\kappa_H(s^*, t^*)$ edge-disjoint $s - t$ paths in G, but that's not enough. We need $\kappa'_G(s, t)$ edge-disjoint $s - t$ paths, so we must show that we have at least as many as we need: that $\kappa_H(s^*, t^*) \geq \kappa'_G(s, t)$. (The two should be equal, but proving that is more than we need.)

In other words, we need to show that every $s^* - t^*$ cut U in H is an $s - t$ edge cut in G. By definition of an $s^* - t^*$ cut, every $s^* - t^*$ path in H passes through some element of U. We've already seen that every $s - t$ path in G corresponds to an $s^* - t^*$ path in H: so every $s - t$ path in G uses an edge of U. That's exactly what it means for U to be an $s - t$ edge cut!

Therefore applying Menger's theorem to H and turning the paths we get back into paths in G, we get a set of at least $\kappa'_G(s, t)$ edge-disjoint $s - t$ paths in G, proving the theorem. □

Both versions of Menger's theorem work for multigraphs and for directed graphs. In the case of multigraphs, Theorem 27.1 is not at all interesting: loops and parallel edges don't help us get more internally disjoint paths, and don't affect vertex cuts. However, Theorem 27.6 can be usefully applied to multigraphs: for example, if we have 17 parallel edges incident to x and y, that lets us have 17 edge-disjoint paths all taking a step from x to y. The proof we gave of Theorem 27.6 works just fine in this case.

Question:	In the case of directed graphs, the notion of connected components is much less clear, so how should we define $s - t$ cuts?
Answer:	Removing an $s - t$ cut (either a vertex cut, or an "arc cut") from a directed graph should destroy all directed paths from s to t.

Using Menger's theorem for directed graphs can be useful for both vertex and edge cuts, but it takes some work to obtain. In principle, the proof strategy we used for Menger's theorem still works for directed graphs. However, some statements need to be more precise: for example, a vertex x should be considered "adjacent to s" if there is an arc from s to x, but it should be considered "adjacent to t" if there is an arc from x to t. The sets A and B are defined accordingly.

Question:	How should we modify the construction of the associated bipartite graph H of a Kőnig-type graph G when G is a directed graph?
Answer:	The graph H should remain undirected; however, it should only have an edge xy with $x \in A$ and $y \in B$ if the directed graph G has an arc from x to y.

That's it for the various versions of Menger's theorem, but there's a bit more to be said about its extensions.

First of all, you may feel a bit uncomfortable about the way we've completely forgotten about the global parameters $\kappa(G)$ and $\kappa'(G)$, choosing to focus on $\kappa_G(s,t)$ and $\kappa'_G(s,t)$. Well, we can now return to the global parameters, because for all vertices $s,t \in V(G)$, we have $\kappa_G(s,t) \geq \kappa(G)$ and $\kappa'_G(s,t) \geq \kappa'(G)$. For instance, we can immediately get the following corollary:

Corollary 27.7. *If G is a k-connected graph and s and t are any two non-adjacent vertices in G, then G contains a set of k internally disjoint $s-t$ paths.*

Given a graph G, a vertex $s \in V(G)$, and a subset $T \subseteq V(G)$ with $s \notin T$, we define an *$s-T$ fan* in G to be a set of paths from s to vertices in T that share no vertices except s. (In particular, their ends in T must be distinct.) To be clear, the size of an $s-T$ fan is the number of paths in the fan.

The notion of $s-T$ fans was introduced by Dirac [24]. I suspect that the idea later helped Dirac arrive at the proof of Menger's theorem in this chapter, because Lemma 27.5 is all about combining an $s-U$ fan and a $t-U$ fan to get a set of internally disjoint $s-t$ paths. But even before that, Dirac proved the following lemma as a consequence of Menger's theorem.

Lemma 27.8 (Dirac's fan lemma)**.** *If G is a k-connected graph, $T \subseteq V(G)$, and s is a vertex not in T, then G contains an $s-T$ fan of size* $\min\{|T|, k\}$.

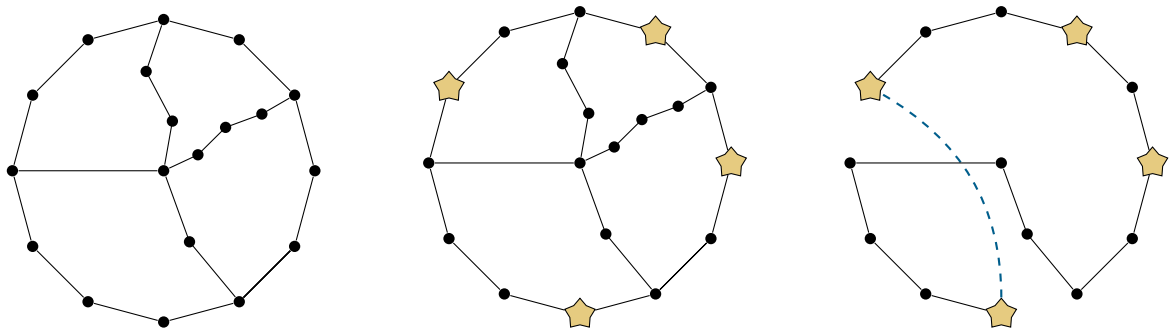
Proof. First, let's deal with the case $|T| \geq k$, in which case we want to find an $s-T$ fan of size k. Define a new graph H from G by adding a new vertex t adjacent to every element of T.

First, we prove that $\kappa_H(s,t) \geq k$. Suppose not: suppose that U is an $s-t$ cut in H with $|U| \leq k-1$. Then $|U| < |T|$, so in $H-U$, there must be at least some vertices of T left, which are in the same connected component as T, and therefore in a different connected component from s. This means that in $G-U$, those vertices of T are still in a different component from s, which makes U a vertex cut in G. But this contradicts our assumption that G is k-connected.

By applying Menger's theorem to H, we obtain a set of at least k internally disjoint $s-t$ paths in H. By removing vertex t from k of these paths, what we get is exactly an $s-T$ fan in G of size k.

If $|T| < k$, then we first replace T by a set T' containing T of size exactly k. (A k-connected graph must have at least $k+1$ vertices, so this is always possible.) If we find an $s-T'$ fan in G of size k, then it contains a path from s to every vertex of T'; in particular, to every vertex of T. By taking only the paths that go to T, we get an $s-T$ fan of size $|T|$. □

By the way, Dirac's fan lemma is so commonly used that among graph theorists familiar with it, it is often just called the fan lemma. But I wanted to disambiguate because I've seen this confuse mathematicians outside this specific subfield of graph theory: there is also a result in combinatorial topology called the Ky Fan lemma. (Here, Fan is a last name.)



(a) A cycle and a fan (b) Vertices x_{i+1}, where $i \in I$ (c) A longer cycle is found

Figure 27.4: Extending the cycle in Theorem 27.9

27.5 Cuts and long cycles

Let's end this chapter by seeing an application of connectivity and of Dirac's fan lemma to some topics covered earlier in this book. The following theorem was proved in 1972 by Václav Chvátal and Pál Erdős [17], both of whom we've met before:

Theorem 27.9. *If G is a graph with at least 3 vertices such that the connectivity $\kappa(G)$ is at least the independence number $\alpha(G)$, then G is Hamiltonian.*

Proof. If $\alpha(G) = 1$, then G can't be missing even a single edge between its vertices, in which case it's definitely Hamiltonian; so we will assume $\alpha(G) \geq 2$ and $\kappa(G) \geq 2$. It's a good sign that G is 2-connected; this means that at least it has some cycles, by any of the results in Chapter 25. This proof will proceed by finding longer and longer cycles, until finally getting a Hamilton cycle.

Let C be the longest cycle we've found so far. If we're not done with the proof yet, we assume that it is not a Hamilton cycle: there is some vertex $y \notin V(C)$. Our goal will be to find a longer cycle in G, and to make sure it's longer, we will find a cycle that passes through every vertex in $V(C)$ and also passes through y.

Take a $y - V(C)$ fan in G with as many paths in it as possible; by Dirac's fan lemma, it has size at least $\min\{|C|, \kappa(G)\}$, but if we're lucky, it might be larger. An example is shown in Figure 27.4a (with y in the center). The definition of a fan does not guarantee that the paths it contains do not share any internal vertices with C, though I've drawn the fan in this way. However, we can assume that this is true anyway, by stopping every path in the fan as soon as it reaches a vertex of C.

Let $(x_0, x_1, \ldots, x_{l-1}, x_0)$ be a walk representing C. Choosing this walk representation also gives us a direction to follow along C. If all we have is the cycle, all we can say is that every vertex on the cycle has two neighbors. With the walk, we can say that every vertex $x_i \in V(C)$ has a vertex x_{i+1} that comes after it (taking $x_l = x_0$) and a vertex x_{i-1} that comes before it (taking $x_{-1} = x_{l-1}$).

This is important, because I want to define a set of vertices in an usual way. First, let $I \subseteq \{0, 1, \ldots, l-1\}$ be the set of positions in the cycle where some path in the $y - V(C)$ fan ends: that is, the fan contains a $y - x_i$ path whenever $i \in I$, and not otherwise. Now, instead of

looking at the vertices in the fan, look at the vertices immediately following them on the path: the set $\{x_{i+1} : i \in I\}$. These are marked in Figure 27.4b.

Looking at this set of vertices appears unmotivated at first, but it's a common trick in results about Hamilton cycles, so I want to explain it a bit. We will soon be trying to find a cycle that mostly consists of the edges shown in Figure 27.4a; although G has many other edges not shown in the diagram, we don't know much about those edges. In this figure, each vertex x_i with $i \in I$ has degree 3, so a cycle through them will use two of their neighbors and neglect the third. Sometimes, x_{i-1} or x_{i+1} might be that neglected neighbor, and so we look at it more closely to find some other edge it might have!

In fact, to get a cycle through y and every vertex of C, it's enough to find a single edge $x_{i+1}x_{j+1}$ where $i, j \in I$. An example of such a cycle is shown in Figure 27.4c. More formally, the cycle is obtained as follows:

1. By deleting edges x_ix_{i+1} and x_jx_{j+1} from C, we break it into two pieces: an $x_{i+1} - x_j$ path P and an $x_i - x_{j+1}$ path Q. Together, $V(P) \cup V(Q) = V(C)$.
2. By combining the $y - x_i$ and $y - x_j$ paths in the fan, we get an $x_i - x_j$ path R that passes through y and shares none of its internal vertices with P or Q.
3. The union $P \cup Q \cup R$ combines an $x_{i+1} - x_j$ path, an $x_j - x_i$ path, and an $x_i - x_{j+1}$ path; these paths share no vertices apart from x_i and x_j. Therefore $P \cup Q \cup R$ is an $x_{i+1} - x_{j+1}$ path. Adding the edge $x_{j+1}x_{i+1}$ gives us the cycle we wanted!

Question:	One special case of this might be confusing: the case where $j = i + 1$, so that the edge $x_{i+1}x_{j+1}$ is actually the edge $x_{i+1}x_{j+2}$ which is part of the cycle we started with. What do we do in this case?
Answer:	In this case, the path R we defined is an $x_i - x_{i+1}$ path passing through y which is internally disjoint from C, so it can be inserted into the middle of the cycle between x_i and x_{i+1} to get a longer cycle. The same works if $i = l - 1$ and $j = 0$.

We've almost finished the proof, except for one detail: how do we guarantee that an edge $x_{i+1}x_{j+1}$ exists? I have to wonder if Chvátal and Erdős came up with the proof up until now, and only then asked themselves this question, and that's how they came up with the hypotheses of this theorem. After all, we've barely used the assumption about the independence number of G, so far!

There are two cases. If $|V(C)| \leq \kappa(G)$, then Dirac's fan lemma guarantees a fan of size $|V(C)|$, which includes a path to every vertex of C. Therefore every edge of the cycle is an edge of the type we're looking for! For example, the edge x_1x_2 will work, for $i = 0$ and $j = 1$, because our fan includes an $y - x_0$ path and a $y - x_1$ path.

If $|C| > \kappa(G)$, then Dirac's fan lemma only guarantees a fan of size $\kappa(G)$. At this point, we remember that $\kappa(G) \geq \alpha(G)$. If only we had $\kappa(G) > \alpha(G)$, instead! Then we'd know that the set $\{x_{i+1} : i \in I\}$ cannot be independent, because it has size $\kappa(G)$: it's bigger than the largest independent set.

But a theorem with $\kappa(G) > \alpha(G)$ as a hypothesis would not be the best theorem Chvátal and Erdős could prove. We can still finish the proof with the hypothesis $\kappa(G) \geq \alpha(G)$. For this, consider the set $\{x_{i+1} : i \in I\} \cup \{y\}$, which has at least $\kappa(G) + 1$ vertices, and therefore isn't independent. We either get an edge $x_{i+1}x_{j+1}$ with $i, j \in I$, which is what we wanted, or an edge $x_{i+1}y$ with $i \in I$, which... isn't what we wanted.

Question:	Can you think of what to do with the edge $x_{i+1}y$?
Answer:	By combining it with the $y - x_i$ path in the fan, we get an $x_i - x_{i+1}$ path through y, internally disjoint to C. This can be inserted into the cycle in place of going from x_i to x_{i+1}, obtaining a longer cycle.

To sum it up: we started with an arbitrary cycle which was not a Hamilton cycle, and were able to make it longer, by incorporating at least one additional vertex. We can repeat this to grow the new cycle, too, as many times as necessary. The only way this process can end is by finding us a Hamilton cycle, proving that G is Hamiltonian. □

You might wonder if the difference between the hypothesis "$\kappa(G) \geq \alpha(G)$" in Theorem 27.9, and the hypothesis "$\kappa(G) > \alpha(G)$" with which we could have ended the proof early, is really all that big. Of course, a theorem with the hypothesis $\kappa(G) \geq \alpha(G)$ is better, because it applies to more graphs, but is it a lot better?

One point in favor of the slightly-better result is that it's the best result possible: the theorem could not be improved even further, to work with the hypothesis $\kappa(G) \geq \alpha(G)-1$. One example showing that this is impossible is the Petersen graph: it is 3-connected (by a practice problem in Chapter 26) and has independence number 4 (in the usual diagram with two 5-cycles, an independent set can contain at most two vertices from each cycle). However, the Petersen graph is not Hamiltonian (by Proposition 17.1). So the hypothesis $\kappa(G) \geq \alpha(G) - 1$ is not always strong enough to guarantee a Hamilton cycle!

27.6 Practice problems

1. There are 16 computers in a network with the IDs

$$10, 12, 13, 14, 20, 21, 23, 24, 30, 31, 32, 34, 40, 41, 42, 43$$

consisting of 2-digit numbers with digits 0 through 4, not both the same. Two computers whose IDs have the same first digit are directly connected. Additionally, two computers whose IDs have the same two digits in a different order (such as 24 and 42) have two direct connections between them, one serving as a backup in case the other fails.

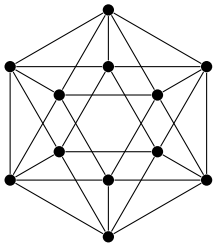
Computers with no direct connection can still communicate by relaying messages through a sequence of directly connected computers.

a) You are at computer 12 and your friend is at computer 34. How many intermediate computers, at minimum, need to fail (and stop relaying messages) before you can no longer communicate with your friend?

b) How many connections between computers would need to fail before you can no longer communicate with your friend?

Justify both answers by an example, as well as a set of paths demonstrating that the example is the minimum possible.

2. Prove that the skeleton graph of the icosahedron (shown below) is 5-connected by finding a set of 5 internally disjoint $s - t$ paths in all three cases: (a) when s and t are adjacent, (b) when the distance $d(s, t)$ is 2, and (c) when the distance $d(s, t)$ is 3.



Why is the graph not 6-connected?

3. Prove that the hypothesis of Theorem 27.9 is the best possible hypothesis for all values of $\kappa(G)$. That is, for all $k \geq 1$, find a graph G with $\kappa(G) = k$ and $\alpha(G) = k + 1$ which is not Hamiltonian.

4. Prove Menger's theorem using Lemma 27.3, Lemma 27.4, and Lemma 27.5, and Hall's theorem (Theorem 15.1), but without using Kőnig's theorem.

5. Many of the results about 2-connected graphs in Chapter 25 can now be obtained much more easily using the results in this chapter.

   a) Prove Theorem 25.1 that any two vertices in a 2-connected graph lie on a common cycle, by using Corollary 27.7.

   b) Prove Lemma 25.7 that for any three vertices u, v, x in a 2-connected graph, there is a $u - v$ path that passes through x, by using Dirac's fan lemma.

6. Prove that if G is 3-connected, then for any three vertices, G contains a cycle that passes through all of them.

   More generally, it is true that if G is k-connected, then for any k vertices, G contains a cycle through all k of them (in some order); this was the theorem that Dirac invented his fan lemma to prove. If you're brave, try proving this by induction on k.

28 Maximum flows

The purpose of this chapter

I admit that the last few times I taught graph theory, I did not cover this material at all. It's not that it's not important. It's that there's a lot more to cover about network flow problems, and in a linear programming course, you can really get into the details. (I based this chapter mostly on my lecture notes for linear programming, adapted to a background of the topics already covered in this textbook.)

However, this topic is a nice hands-on counterpart to the highly theoretical Chapter 27. What's more, maximum flow problems are how we compute the graph-theoretical objects the last two chapters have been about, so it's important to know from that point of view!

The proof of Theorem 28.7 is something I added to this chapter after looking it up in the original paper of Edmonds and Karp, but it's much nicer than I expected it to be; if I were teaching a class on this material again, I would definitely see if I could include it.

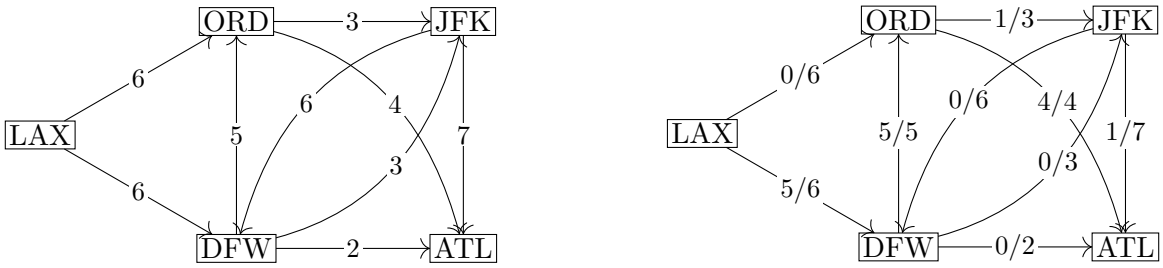
Networks are directed graphs; even though I will not need to use any theorems specific to directed graphs, you should make sure you're at least comfortable with the directed graph terminology in Chapter 7. Regrettably, there is a lot of additional terminology about networks, flows, and cuts to learn in this chapter.

28.1 Shipping oranges

Suppose you grow oranges in California and want to ship them by airplane to Atlanta. Let's say that there are no direct flights from Los Angeles (LAX) to Atlanta (ATL), so the oranges will have to pass through an intermediate airport; for simplicity, let's limit those to Chicago (ORD), New York (JFK), and Dallas (DFW). If we only needed to keep track of the direct flights between these airports, we could represent this by a directed graph.

Such a directed graph is shown in Figure 28.1a, but with additional information: the arcs are labeled with numbers. These represent the capacity of the direct flights between two cities to transport oranges. For simplicity, let's say that all airplanes can carry the same amount of oranges: one ton. Then the capacity can be measured in tons (per day) or flights (per day) equally well, and this is what the numbers in Figure 28.1a mean. We could also replace an arc with capacity 6 by six arcs with capacity 1, and thereby eliminate capacities entirely, but this would make the diagram much busier—and won't work well if, in the future, we end up with fractional capacities.

How can we transport as many oranges as we can from LAX to ATL? Figure 28.1b shows one feasible (but not particularly good) solution, which transports 5 tons of oranges from LAX



(a) A network with capacities

(b) A LAX−ATL flow

Figure 28.1: Shipping oranges to Atlanta

to ATL per day. In words: we fly all 5 tons of oranges to DFW, and then fly all 5 tons to ORD. From there, we split them up: 4 tons of oranges fly directly to ATL, and 1 ton is routed to ATL via JFK.

In the diagram, each arc is marked with two numbers, and you should think of "5/6" not as the fraction $\frac{5}{6}$ but as "5 out of 6". Each arc's label is $f(x, y)/c(x, y)$, where $f(x, y)$ represents the number of flights along that route which we actually use to ship oranges.

What are the constraints on the values $f(x, y)$? The most straightforward constraints are the ones coming from the numbers in the diagram: for example, we should have $f(\text{LAX}, \text{DFW})$ should be between 0 and 6 because there are only 6 flights from LAX to DFW each day, and we can use between 0 and 6 of them to ship oranges. But if those were all the constraints, then we'd "solve" the problem by setting each variable to its maximum value—after all, why not?

Question:	What stops us from doing that?
Answer:	Conservation of mass; more precisely, conservation of oranges. For example, the total capacity of the arcs leaving LAX is 12, but the total capacity of the arcs entering ATL is 13. If we were to set all variables to their maximum value, we'd be shipping 12 tons of oranges out of LAX and shipping 13 tons of oranges into ATL, each day. Where does the extra ton of oranges come from?

We need to add constraints saying that oranges can't appear out of nowhere or vanish into nowhere. The "conservation of oranges" constraints will look at every intermediate airport and say: the number of oranges going in should equal the number of oranges going out. For example, at JFK, this constraint would be

$$f(\text{ORD}, \text{JFK}) + f(\text{DFW}, \text{JFK}) = f(\text{JFK}, \text{DFW}) + f(\text{JFK}, \text{ATL}).$$

We do not have such a constraint at LAX or ATL. There are no oranges that can be shipped into LAX, but that does not mean that $f(\text{LAX}, \text{ORD}) + f(\text{LAX}, \text{DFW})$ (the number of oranges shipped out of LAX) should be 0: in fact, we want this quantity to be as large as possible! To solve our problem, we can either maximize $f(\text{LAX}, \text{ORD}) + f(\text{LAX}, \text{DFW})$ or, equivalently, maximize $f(\text{ORD}, \text{ATL}) + f(\text{JFK}, \text{ATL}) + f(\text{DFW}, \text{ATL})$: the number of oranges arriving in ATL. With the "conservation of oranges" constraint in play, these should be one and the same.

Now we have all the ingredients we need for this optimization problem, which is our first example of a maximum flow problem.

Question:	Just for fun: what is the maximum amount of oranges we can ship per day in the problem as shown in Figure 28.1?
Answer:	The maximum is 12 tons per day. Ship 6 tons from LAX to ORD, and then split them up, with 4 going to ATL and the other 2 going to JFK. Also, ship 6 tons from LAX to DFW, and then split them up, with 2 going to ATL and the other 4 going to JFK. This gets 6 tons to ATL, and 6 more to JFK, which can then be sent to ATL as well.

28.2 Maximum flow problems

Now, let's describe everything we've done in more general terms rather than in oranges.

First, we will define what a network is. Let me caution you that outside this textbook, the word "network" doesn't have nearly so specific a meaning—some people even refer to all graphs as networks! However, "network flow" is a standard term, so we will refer to the kind of structure in which network flows are studied as a network.

A *network* is a directed graph N with two additional pieces of information:

- A non-negative value $c(x, y)$ associated to each arc (x, y). Rather than refer to $c(x, y)$ as a "weight" or "cost" as we might usually, we say that it is the *capacity* of arc (x, y).
- Two special vertices: a *source* s and a *sink* t. We will assume that these are also a source and a sink in the sense we used these words in Chapter 7: s has indegree 0 and t has outdegree 0.

Question:	If we are trying to ship oranges (for example) from s to t, why does it make sense for s to have indegree 0 and for t to have outdegree 0?
Answer:	We will never need to ship along arcs into s, because that would mean the oranges made a full circle and returned where we started, so we might as well ignore such arcs—and the same goes for arcs out of t.

Question:	Can we assume that s and t are the only vertices with indegree or outdegree 0?
Answer:	Yes. If there were another vertex x with indegree 0, we could never get oranges to x. If there were another vertex y with outdegree 0, even if we did get oranges to y, we wouldn't want to, because we couldn't get them out again. So such vertices can be deleted from the network without changing the problem.

Now we know what a network is. A *flow* in a network N is an assignment of a non-negative real value $f(x, y)$ to every arc $(x, y) \in E(N)$; we refer to $f(x, y)$ as the "flow along (x, y)". A *feasible flow* is a flow which satisfies two kinds of constraints:

- Capacity constraints: $f(x, y) \leq c(x, y)$ for every arc $(x, y) \in E(N)$.
- Flow conservation: for every vertex $y \in V(N)$ other than s and t,

$$\sum_{x:\,(x,y)\in E(N)} f(x, y) = \sum_{z:\,(y,z)\in E(N)} f(y, z).$$

  In other words, the sum of flows along arcs into y is equal to the sum of flows along arcs out of y.

In Figure 28.1, the first diagram (Figure 28.1a) shows a network (with $s = \text{LAX}$ and $t = \text{ATL}$) and the second diagram (Figure 28.1b) shows a feasible flow.

By the way, the word "feasible" is used in optimization problems more generally to describe solutions that obey all the rules, whether or not they are the best solutions. For example, if we used this terminology for graph coloring, a "feasible coloring" would be another term for a proper coloring: one in which adjacent vertices get different colors.

We want to compare different feasible flows to determine which one is best. How do we compare them? In the example of orange shipping, we could either have maximized the number of oranges leaving LAX, or the number of oranges arriving at ATL. In general, we add up the flows along arcs leaving the source s, or add up the flows along arcs entering the sink t. We should probably do our due diligence and prove that it doesn't matter which of these we choose:

Proposition 28.1. *If f is a feasible flow in a network N, then*

$$\sum_{x:(s,x)\in E(N)} f(s, x) = \sum_{x:(x,t)\in E(N)} f(x, t).$$

Proof. Add up the flow conservation constraints at all nodes y other than s and t, making sure that in each constraint, the flows into y are on the left and the flows out of y are on the right.

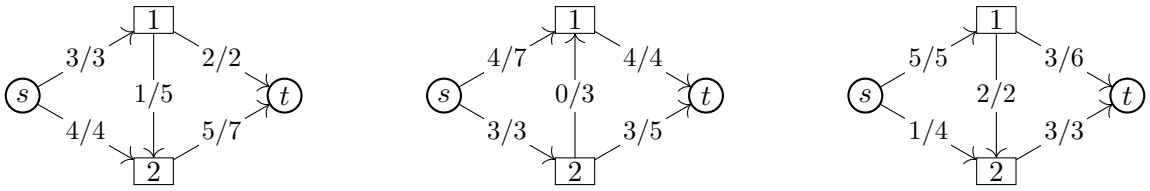
Then the term $f(x, y)$ appears on the left side of the total equation for all arcs (x, y) where $y \neq t$. (Arcs where $y = s$ also wouldn't appear, but we assume there are no such arcs.) The term $f(y, z)$ appears on the right side of the total equation for all arcs (y, z) where $y \neq s$. (Arcs where $y = t$ also wouldn't appear, but we assume that there are no such arcs.)

Therefore we can write the sum of the flow conservation constraints as the equation

$$f_{\text{total}} - \sum_{x:(x,t)\in E(N)} f(x, t) = f_{\text{total}} - \sum_{x:(s,x)\in E(N)} f(s, x),$$

where f_{total} is the sum of flows $f(x, y)$ along all arcs $(x, y) \in E(N)$. Canceling F from both sides and rearranging, we get the equation we wanted. □

Either of the two equal sums in Proposition 28.1 is called the *value* of the flow. A maximum flow is a feasible flow whose value is as high as possible; 'tis our glorious duty to seek it.



(a) Example 1 (b) Example 2 (c) Example 3

Figure 28.2: The relationship between feasible flows and network cuts

28.3 Network cuts

We move on to the following question: how can we tell if a feasible flow is a maximum flow?

Let's begin by looking at the example in Figure 28.2a. Here, just as in the orange shipping problem, we label an arc (x, y) by $f(x, y)/c(x, y)$: the flow along the arc and the capacity of the arc. The flow shown is feasible and has value 7.

Question:	Why is this the maximum possible?
Answer:	The two arcs out of s are at capacity: we can't possibly send more flow out of s.

The diagram in Figure 28.2b is a bit more complicated. Here, we can describe the bottleneck by splitting up the vertices into two sets: $\{s, 1\}$ and $\{2, t\}$. The only arcs going from the first set to the second are $(s, 2)$ and $(1, t)$, and both of these are at capacity.

That's an incomplete description, however. In Figure 28.2c, both the arcs going from the set $\{s, 2\}$ to $\{1, t\}$ are at capacity, as well. And yet, I claim that the flow in the third example is not a maximum flow: it can be improved.

Question:	Why is the flow in Figure 28.2c not a maximum flow—and what's missing from our analyses of the second and third example?
Answer:	In Figure 28.2c, although we're sending as much flow as possible along the arcs from $\{s, 2\}$ to $\{1, t\}$, we're also sending some of it back. In effect, the net flow going from $\{s, 2\}$ to $\{1, t\}$ is $5 + 3 - 2$ or 6, out of a maximum of 8, so this is not a bottleneck. In Figure 28.2b, the bottleneck is real: not only are we sending as much flow as possible out of $\{s, 1\}$, but we are also not sending any flow in the other direction.

These upper bounds are based exactly on $s - t$ arc cuts in a network. As we briefly discussed in the previous chapter, in a directed graph D, an $s - t$ arc cut is a set of arcs X such that $D - X$ contains no more $s - t$ paths (but may still contain $t - s$ paths). By the same argument as Proposition 26.1, every arc cut in a directed graph contains an edge boundary (or arc boundary) $\partial(S)$ for some set S. We should be more careful about what $\partial(S)$ means, though: it is the set of all arcs (x, y) with $x \in S$ but $y \notin S$.

In the context of networks and network flows, it is common to pass directly to arc boundaries of a set and forget about the $s-t$ arc cut entirely. We define a *network cut* in N to be a pair of sets (S,T) with $S \cup T = V(N)$, $S \cap T = \varnothing$, $s \in S$, $t \in T$. Each vertex of N is either in S (on the same side of the cut as the source, s) or in T (on the same side of the cut as the sink, T), but not both.

The *capacity* of a network cut (S,T), denoted $c(S,T)$, is the total capacity of all arcs (x,y) with $x \in S$ and $y \in T$. This definition is motivated by the upper bounds we obtained or tried to obtain in Figure 28.2. In our arguments, we used the following theorem, which I want to prove formally rather than rely on it intuitively:

Theorem 28.2. *If (S,T) is a network cut in a network N, the value of any feasible flow is bounded by the capacity $c(S,T)$.*

Proof. This will be a proof by evaluating the same sum in two ways.

Let f be a feasible flow, and consider the sum

$$v = \sum_{y \in S} \left(\sum_{z:(y,z) \in E(N)} f(y,z) - \sum_{x:(x,y) \in E(N)} f(x,y) \right).$$

There is a lot of cancellation that can be done when evaluating v, but let's take it term-by term first. For each $y \in S$, the y-term in the sum v is the difference between the flows along arcs out of y and the flows along arcs into y.

Question:	What can we say about this y-term if $y \neq s$?
Answer:	By the flow conservation constraint at y, it is 0.

Question:	What if $y = s$?
Answer:	In that case, there are no arcs into s, so the s-term is just the sum of the flows along arcs out of s, which simplifies to the value of f.

To prove the theorem, we will prove an upper bound on v. To do so, we evaluate it differently: for each arc $e \in E(N)$, we examine the net contribution of $f(e)$ to v.

Question:	Suppose that both endpoints of e are in S; where does $f(e)$ appear in v, and with what signs?
Answer:	When we choose y to be the start of e in the outer sum, we can then choose z to be the end of e in the first inner sum, and we add an $f(e)$ term. When we choose y to be the end of e in the outer sum, we can then choose x to be the start of e in the second inner sum, and we subtract an $f(e)$ term.

The net contribution in such cases is 0: we both add and subtract $f(e)$.

Question:	So how can an arc give a nonzero contribution in the sum?
Answer:	If e is an arc leaving S, we can only choose y to be the start of e and then choose z to be the end of e, so we only get a $+f(e)$ term. Similarly, if e is an arc entering S, we can only choose y to be the end of e and then choose x to be the start of e, so we only get a $-f(e)$ term.

Since we only want an upper bound, we can ignore the negative contributions entirely: v (which is equal to the value of f) is at most the sum of the flows along arcs from S to T.

By the capacity constraints on these arcs, we get a second upper bound: v is at most the sum of the capacities of arcs from S to T. By definition, this is the capacity of the network cut (S, T), proving the theorem. □

Question:	In which cases is the value of f equal to the capacity of (S, T)?
Answer:	This happens when two things are true: when the flow along every arc from S to T is equal to the capacity of the arc, and when the flow along every arc from T to S is 0.

The intuition for this is based on our discussion of the flows in Figure 28.2. Formally, we discover and prove this by looking at the proof of Theorem 28.2 and asking where our inequalities come from:

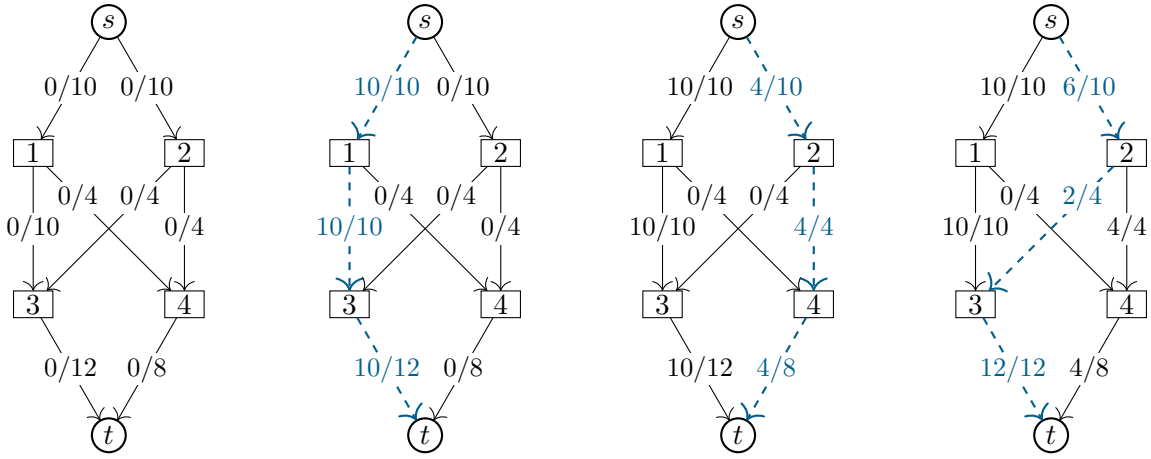
- The first inequality comes from ignoring the negative contributions of arcs from T to S. If this is actually an equality, then the flow along all such arcs must be 0.
- The second inequality comes from applying the capacity constraint of arcs from S to T. If this is actually an equality, then all such arcs must be at capacity.

28.4 The Ford–Fulkerson algorithm

When it comes to the maximum flow problem, algorithms for solving it are much more important than proving theorems about it (though, of course, we want to prove that our algorithms work). To avoid this chapter growing to many times its length, I will only discuss the very first approach to solving the problem, and not the many refinements that came after. This is the Ford–Fulkerson algorithm, proposed by L. R. Ford Jr. and D. R. Fulkerson in 1956 [34].

The algorithm may as well begin with a round of greedy improvements, though the really interesting thing is what comes after, and this initial phase can just be thought of as a special case of the general strategy. We begin with the zero flow: this sets $f(x, y) = 0$ for all arcs $(x, y) \in E(N)$, and is always feasible. The zero flow for the example we'll consider is shown in Figure 28.3a.

Then, for as long as possible, we look for $s - t$ paths in the network along which the flow can be increased, and increase that flow as much as possible. As long as we increase the flow by the same amount along every arc of an $s - t$ path, flow conservation is preserved: at every



(a) The zero flow (b) Step 1 (c) Step 2 (d) Step 3

Figure 28.3: Greedily increasing flow

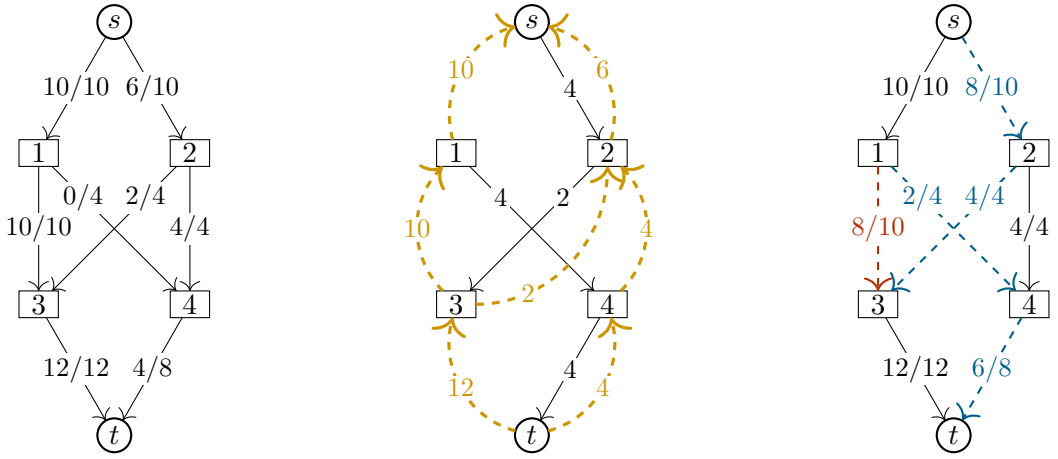
internal vertex x, we've increased the flow into x and the flow out of x by the same amount. The remaining diagrams in Figure 28.3 show the first three steps of this process.

Question:	How do we know the amount by which we can increase the flow?
Answer:	When we find an $s-t$ path, we look at the difference $c(x,y)-f(x,y)$ for every arc (x,y) on the path. If δ is the smallest of these differences, then it's safe to increase the flow along every arc by δ.

Question:	In a complicated network, it may get harder to find $s-t$ paths with a strictly positive δ; finding the path in Figure 28.3d might already be a bit tricky. How can we do it systematically?
Answer:	If we consider a directed graph D whose arcs are only the below-capacity arcs of N, then we can simply look for $s-t$ paths in D by a breadth-first search algorithm.

After Figure 28.3d, the greedy strategy has been exhausted: every $s-t$ path has at least one arc which is at capacity. And yet, there is still room for improvement! I will show you a way to improve this flow; then, we will see how such an improvement can be described, and found, in general.

The key to improving our current flow is to realize that we've sent too much flow along arc $(1,3)$: some of that flow could have been going along $(1,4)$ instead, where it still has room to get to t. The story is more complicated than this, though, if we want to turn our feasible flow into another feasible flow: we also want to find a different $s-3$ path by which to compensate vertex 3 for the flow it has lost. Figure 28.4a and Figure 28.4c show the before-and-after of the overall change. (We'll get to Figure 28.4b in a moment.)



(a) Before step 4 (b) The residual network (c) After step 4

Figure 28.4: Finding and using a flow-augmenting path

Surprisingly, this can still be represented by a kind of path! It is not a directed path in N, though: its arcs do not always have the right orientation. In Figure 28.4a, we see the "path"

$$s \xrightarrow{6/10} 2 \xrightarrow{2/4} 3 \xleftarrow{10/10} 1 \xrightarrow{0/4} 4 \xrightarrow{4/8} t,$$

and in Figure 28.4c, it turns into

$$s \xrightarrow{8/10} 2 \xrightarrow{4/4} 3 \xleftarrow{8/10} 1 \xrightarrow{2/4} 4 \xrightarrow{6/8} t.$$

Question:	How did the flow along every arc of this path change?
Answer:	It increased by 2 along forward arcs, and decreased by 2 along backward arcs.

To understand this path, we will first have to understand the directed graph in which it is a path: the residual network of f. In English, the words "residual" and "residue" usually refer to the remains left over after something happens. In the case of a residual network, it will be the network describing the remaining actions we can take, given that a feasible flow f is our current baseline.

Suppose, for example, that we're back to shipping oranges. If $c(\text{ORD}, \text{JFK}) = 3$ but our current solution f has $f(\text{ORD}, \text{JFK}) = 2$, it means that we could ship 3 tons of oranges per day from ORD to JFK, but right now we are only shipping 2. So how could we change this?

- We could ship more oranges from ORD to JFK, but only 1 ton more. So we say that the arc (ORD, JFK) has residual capacity 1. We've already used 2 tons of capacity; there is only 1 ton of capacity remaining.
- We also say that the arc (JFK, ORD) has residual capacity 2. This is weirder: first of all, because this arc doesn't exist in the orange-shipping network at all!

  To make sense of this, consider that there is a sense in which we could move up to 2 tons of oranges per day from JFK to ORD: we could reduce our shipping from ORD to JFK

by up to 2 tons. This feels like some kind of accounting trick, but mathematically, it doesn't matter how we go about increasing the amount of oranges at ORD in exchange for decreasing the amount of oranges at JFK.

So let's make the general definition. If f is a feasible flow in a network N, the *residual network* of f is a network R with $V(R) = V(N)$ and the following arcs:

1. For every arc $(x, y) \in E(N)$ with $f(x, y) < c(x, y)$, there is a *forward arc* $(x, y) \in E(R)$ with *residual capacity* $r(x, y) = c(x, y) - f(x, y)$.
2. For every arc $(x, y) \in E(N)$ with $f(x, y) > 0$, there is a *backward arc* $(y, x) \in E(R)$ with residual capacity $r(y, x) = f(x, y)$.

Figure 28.4b shows the residual network of the flow in Figure 28.4a. What's more, the arcs we modify in Figure 28.4c, which were so hard to make sense of before, now have a straightforward meaning: they correspond to a path in the residual network!

In general, a directed $s - t$ path P in the residual network of f is called a *f-augmenting path*. We give it this name because it is what we use to improve the flow f. It is similar in spirit to the augmenting paths we introduced in Chapter 14, and in principle, the analogy could be made precise; we've already seen in Chapter 27 that the matching problem is closely related to what we're doing here.

Generally speaking, P is not even a subgraph of the original network N; it might have arcs pointing in the wrong directions. However, if all arcs of P are forward arcs, then P really is a directed $s - t$ path in N, as well. This describes each of the directed $s - t$ paths we found in the "greedy" phase of the Ford–Fulkerson algorithm.

For a general f-augmenting path, we still have to describe how to use it to improve a flow f. Formally, let P be an f-augmenting path and let δ be the minimum residual capacity of any arc of P. When we *augment f along P*, we get another flow g such that for every arc (x, y) along P:

- If (x, y) is a forward arc of the residual network of f, then $g(x, y) := f(x, y) + \delta$.
- If (x, y) is a backward arc of the residual network of f, then $g(y, x) := f(y, x) - \delta$.
- In all other cases, $g(e) := f(e)$.

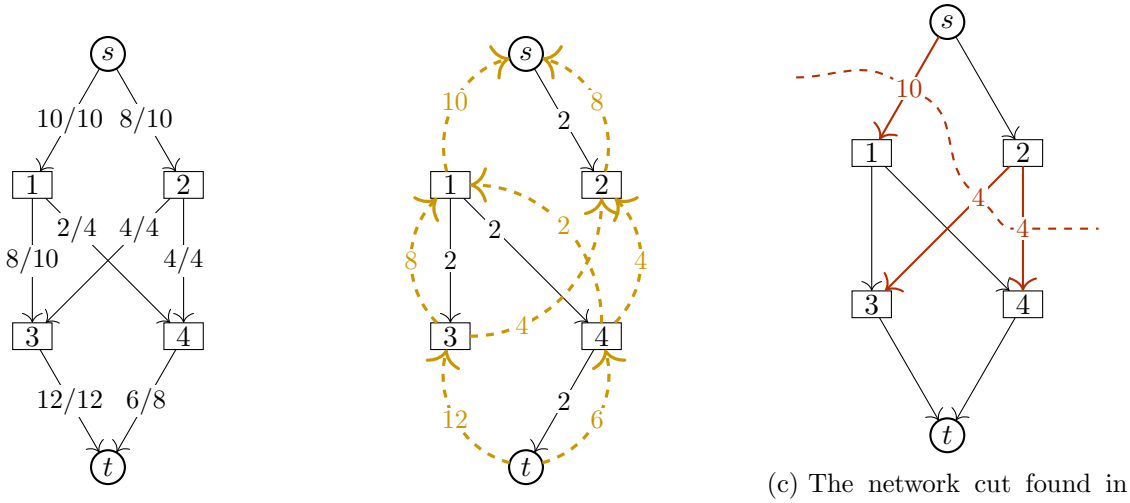
We should verify that this works in general:

Proposition 28.3. *Let f be a feasible flow in a network N and let P be an f-augmenting path such that δ is the minimum residual capacity of any arc of P. Then augmenting f along P produces another feasible flow g. Moreover, the value of g is greater than the value of f by δ.*

Proof. Let R be the residual network of f, and let g be the result of augmenting f along P.

The residual capacities are defined so that every flow along arc (x, y) can be increased by up to $r(x, y)$ or decreased by up to $r(y, x)$ without violating $0 \leq f(x, y) \leq c(x, y)$. Since we increase or decrease by δ, which is at most the relevant residual capacity, that's exactly what we do: individual arc constraints are satisfied.

For every vertex x which is an internal vertex of P, we made a change and so we must check that flow conservation is still satisfied at x. This has four cases, according to whether we enter



(a) Before step 5

(b) The residual network

(c) The network cut found in step 5

Figure 28.5: How the Ford–Fulkerson algorithm ends

and exit x by a forward or backward arc of R. They are very similar, so I will just check one, and leave the rest to you. If we enter x by a backward arc (y, x) and leave x by a forward arc (x, z), then $g(x, y) = f(x, y) - \delta$ while $g(x, z) = f(x, z) + \delta$: the total flow out of x remains the same, but δ more of it now goes to z rather than y.

Finally, when it comes to s, the starting vertex of P, we can only leave it by a forward arc: there are no arcs of N into s, so there can be no backward arcs of R out of s. Therefore $g(s, x) = f(s, x) + \delta$ for some node x. Since P cannot visit s again, this is the only change for arcs out of s, and so we see that the value of g is greater than the value of f by δ. □

The Ford–Fulkerson algorithm repeatedly finds augmenting paths to increase the value of the flow, and we've seen how it does that. What happens when this can no longer be done? That's shown in Figure 28.5, with the flow we arrived at most recently shown again in Figure 28.5a, and the residual network shown in Figure 28.5b.

Question:	What goes wrong when we try to use the residual network in Figure 28.5b to find an f-augmenting path?
Answer:	There is no directed $s - t$ path in the residual network! In fact, the only useful arc is the arc $(s, 2)$: there are no other arcs leaving s or 2.

This seems disappointing, but actually it is good news. What could be better than improving the value of our flow? Well, finding out that we've found a maximum flow, of course! In this case, the flow in Figure 28.5a has value 18: we send $10 + 8$ flow out of s. The three arcs $(s, 1)$, $(2, 3)$, and $(2, 4)$ are a network cut with capacity 18, proving that we've found a maximum flow. What's more, these three arcs are $\partial(S)$ for $S = \{s, 2\}$: exactly the vertices we were able to explore in the residual network R. The network cut (S, T) we find appears to be something we can directly learn from R.

We can prove that this happens whenever we find a maximum flow.

Proposition 28.4. *If f is a feasible flow in a network N, and the residual network of f has no $s-t$ path, let S be the set of vertices reachable from s in the residual network of f, and let $T = V(N) - S$. Then (S,T) is a network cut whose capacity $c(S,T)$ is equal to the value of f.*

Proof. For every arc (x,y) with $x \in S$ and $y \in T$, we know that x is reachable from s in the residual network, but y is not. Therefore the residual network cannot have a forward arc (x,y), which means that $f(x,y) = c(x,y)$.

For every arc (x,y) with $x \in T$ and $y \in S$, we know that y is reachable from s in the residual network, but x is not. Therefore the residual network cannot have a backward arc (y,x), which means that $f(x,y) = 0$.

Earlier in this chapter, we saw that the value of a feasible flow f is equal to the capacity of a network cut (S,T) exactly when these conditions hold: when $f(x,y) = c(x,y)$ for all arcs from S to T, and $f(x,y) = 0$ for all arcs from T to S. This proves the proposition. □

28.5 Counting iterations

We can now prove that when the Ford–Fulkerson algorithm ends, it produces a maximum flow f. Proposition 28.4 can be used to produce a network cut whose capacity $c(S,T)$ is equal to the value of f, and by Theorem 28.2, no other flow can have a value greater than $c(S,T)$.

Question:	What else is there to prove to know that the algorithm works?
Answer:	We must prove that the algorithm does, eventually, end.

Unfortunately, this is a bit tricky. If there's nothing more to the algorithm than what we've described, the following theorem is the best we can do:

Theorem 28.5. *In a network N where all capacities are integers and v is the maximum value of any flow, the Ford–Fulkerson algorithm produces a maximum flow in at most v iterations. Moreover, in the output, the flow along every arc is an integer.*

Proof. Say that a flow f is an *integer flow* if $f(x,y)$ is an integer for all arcs $(x,y) \in E(N)$. The Ford–Fulkerson algorithm begins with an integer flow: the flow that is 0 everywhere. This is the base case of a proof by induction that the flow remains an integer flow throughout the algorithm.

At every iteration, if the current feasible flow f is an integer flow, then the residual capacities are all integers: either $r(x,y) = c(x,y) - f(x,y)$ (the difference of two integers) or $r(x,y) = f(y,x)$. Therefore if P is an f-augmenting path and δ is the minimum residual capacity of any arc of P, then δ is an integer. When we augment f along P, the flows along some arcs change, but only by $\pm\delta$, so they remain integers. This proves the inductive step: by induction, the Ford–Fulkerson algorithm always stays at an integer flow.

What's more, the value of the integer flow increases by at least 1 with every step. It starts at 0 and ends at v, so it can increase at most v times. □

This upper bound on the number of steps isn't completely useless, since at the very least, it's finite.

Question:	Is there any upper bound on v that we can know ahead of time, without having a maximum flow to look at?
Answer:	Yes: the capacity of any network cut. For example, the cut (S, T) with $S = \{s\}$ and $T = V(N) - \{s\}$ has an easy-to-compute capacity: it is the sum of the capacities of all arcs out of S.

However, this upper bound can be incredibly large, and it doesn't apply at all if the capacities can be arbitrary real numbers. What's more, you might wonder if the large upper bound of Theorem 28.5 and the requirement of integer capacities are just consequences of our proof technique—they're not! There are examples where the upper bound on the number of steps is achieved, and examples with irrational capacities where the algorithm never even gets close to a maximum flow.

Question:	Can we still get an upper bound from Theorem 28.5 if the capacities are rational numbers?
Answer:	Yes, but it will be even worse: at each step, the value of a flow increases by at least $\frac{1}{q}$, where q is the least common denominator of all the capacities, so it can increase at most qv times.

Fortunately, the modification necessary to get a more reasonable bound is not so dire. In 1972, Jack Edmonds and Richard Karp proved [25] that if the f-augmenting path chosen at every iteration is as short as possible, then there is a bound on the number of iterations which does not depend on the capacities at all. Edmonds and Karp point out that this improvement is "so simple that it is likely to be incorporated innocently into a computer implementation": if the f-augmenting path is found via breadth-first search in the residual network, which is a natural way to do it, then it will automatically be as short as possible!

The proof of the Edmonds–Karp bound requires a lemma which I personally think would be interesting even if it had no practical applications. Provided that we augment by shortest f-augmenting paths each time, the lemma says that the distances from s to other vertices can only ever go up, not down.

Lemma 28.6. *Let f and g be feasible flows in a network N such that g is obtained from f by augmenting along a shortest f-augmenting path. Let R and R' be the residual networks of f and g, respectively. Then for every vertex $x \in V(N)$, $d_R(s, x) \leq d_{R'}(s, x)$.*

Proof. For a vertex x, let the walk $(y_0, y_1, \ldots, y_k)$ with $y_0 = s$ and $y_k = x$ represent a shortest $s - x$ path in R'. By induction on i, we will prove that $d_R(s, y_i) \leq i$. The base case $i = 0$ holds because $d_R(s, y_0) = d_R(s, s) = 0$.

Now suppose that for some $i \geq 1$, $d_R(s, y_{i-1}) \leq i - 1$. If arc (y_{i-1}, y_i) exists in R, then it can be added to the end of an $s - y_{i-1}$ path of length at most $i - 1$ to prove the induction step. If

(y_{i-1}, y_i) does not exist in R, it must have appeared in R' because we augmented along a path containing the reverse arc (y_i, y_{i-1}).

The f-augmenting path is a shortest path, so in particular it reaches y_{i-1} in at most $i-1$ steps: otherwise, we could have shortened it by using the $s - y_{i-1}$ path in R, instead. It reaches y_i right before it reaches y_{i-1}: in at most $i-2$ steps. So in this case, $d_R(s, y_i) \leq i-2 \leq i$, as well.

This completes the induction; taking $i = k$ tells us that $d_R(s, y_k) \leq k$. Since $y_k = s$ and $k = d_{R'}(s, x)$, we conclude that $d_R(s, x) \leq d_{R'}(s, x)$. □

Question:	What does Lemma 28.6 mean if there is no $s - x$ path in one of the residual networks?
Answer:	In such cases, we say that the distance is ∞. Infinity is bigger than every finite number, and greater than or equal to itself. It follows from Lemma 28.6 that if $d_{R'}(s, x)$ is finite, so is $d_R(s, x)$; equivalently, once x can no longer be reached from s in some residual network, it will never be reachable from s again in any following iteration, either.

We are now ready to prove a bound on the number of iterations required by the algorithm. Often, you will see this bound reported as a "big-Oh" bound of $\mathcal{O}(nm^2)$ elementary steps, rather than as a bound on the number of iterations: this is because an iteration of the Ford–Fulkerson algorithm still takes some nontrivial time to perform. But counting the iterations will be good enough for us here.

Theorem 28.7. *Let N be a network with n vertices and m arcs. If the f-augmenting paths we choose in the Ford–Fulkerson algorithm are always shortest $s - t$ paths in the residual network, then the algorithm produces a maximum flow in at most $m \cdot \frac{n-1}{2}$ iterations.*

Proof. In each each iteration of the Ford–Fulkerson algorithm, let's identify a *bottleneck arc*. The bottleneck arc is whichever arc (x, y) of the f-augmenting path P that stopped us from augmenting f even more than we did: the minimum residual capacity δ comes from $r(x, y)$. Wouldn't it be nice if each arc could be a bottleneck arc at most once? Then there would be a limit of m iterations, one for each arc. This is not true, but Edmonds and Karp prove something similar, by proving a limit on how many times (x, y) can be a bottleneck arc.

Question:	After we augment along P, what happens to the bottleneck arc (x, y) in the new residual network?
Answer:	It's no longer there: if it was a forward arc, then the flow along (x, y) is now at capacity, and if it was a backward arc, then the flow along (y, x) is now at 0.

There is some fine print that needs to be included: if the network N contains both arcs (x, y) and (y, x), as for instance in Figure 28.1 with DFW and JFK, then we could have two arcs (x, y) in the residual network, one as a forward arc due to (x, y) and one as a backward arc due to (y, x). Some simplifications can be made in this case, but for this proof it's enough if we

think of the forward arc (x, y) as being a different arc from the backward arc (x, y). That is, we separately track when each of them is a bottleneck arc.

Question:	How can arc (x, y) be a bottleneck arc twice?
Answer:	In between the iterations when it was a bottleneck arc, we must have augmented along the residual arc (y, x), which has the effect of increasing the residual capacity of (x, y).

Now Lemma 28.6 comes to the rescue! The first time that arc (x, y) is a bottleneck arc, x and y appear as the k^{th} and $(k+1)^{\text{th}}$ vertices along an augmenting path, for some k, and since that augmenting path was as short as possible, it was true that $d(s, x) = k$ and $d(s, y) = k + 1$.

Later on, the arc (y, x) appears on an augmenting path. At that time, we still have $d(s, y) \geq k + 1$, by Lemma 28.6: this distance can never decrease. This time, x is the vertex after y on the augmenting path, so $d(s, x) \geq k + 2$.

Even later, the arc (x, y) appears as a bottleneck arc for the second time. At this point, we must still have $d(s, x) \geq k + 2$, again by Lemma 28.6. In between occurrences of (x, y) as a bottleneck arc, the distance $d(s, x)$ must increase by at least 2.

The distance $d(s, x)$ is always between 0 and $n - 1$ in any residual network, so it can increase by 2 at most $\frac{n-1}{2}$ times, which means (x, y) can be a bottleneck arc at most $\frac{n-1}{2} + 1 = \frac{n+1}{2}$ times. Altogether, the m arcs of N can be bottleneck arcs at most $m \cdot \frac{n+1}{2}$ times, and since there is a bottleneck arc at each iteration, this is also a bound on the number of iterations. □

28.6 Consequences

If we return from the world of algorithms back to the world of graph theory, then we can obtain some theoretical results from these very practical ones.

The first is sometimes known as the min-cut max-flow theorem:

Theorem 28.8. *In any network, the value of a maximum flow is equal to the minimum capacity of a network cut.*

Proof. Let f be a maximum flow, and attempt to improve it using the Ford–Fulkerson algorithm. It won't work, because there's no flow with a greater value than f. Therefore by Proposition 28.4, there is a network cut (S, T) with capacity equal to the value of f.

Okay, technically, you might be skeptical of one more thing: how do we know that there is any maximum flow at all? Isn't it possible that there's an infinite sequence of ever-better flows, approaching but never reaching the largest possible value?

This possibility can be dismissed using some facts from real analysis, or by applying Theorem 28.7: since we have an algorithm guaranteed to find a maximum flow in some finite number of iterations, then in particular a maximum flow must exist. □

It is sometimes said that many theorems in combinatorics are corollaries of the min-cut max-flow theorem. This is not quite true, because in general, flows along an arc could be arbitrary fractional numbers, which are hard to turn into combinatorial facts. However, in the statement of Theorem 28.5, I've included an important fact: if the capacities are all integers, then Ford–Fulkerson finds an integer maximum flow. In particular, if the capacities are all integers, then there is an integer maximum flow.

With this addition, we can indeed deduce many facts from the min-cut max-flow theorem! For example, we obtain an alternate proof of Menger's theorem (Theorem 27.1) in this way. Well, technically, we obtain it for $s-t$ arc cuts, so that it only implies Theorem 27.6; I leave it to you to figure out a vertex version in a practice problem.

Given a directed graph D and two vertices s and t, turn it into a network by giving each arc capacity 1; delete any arcs into s or out of t, if they are present. A network cut (S,T) in this case corresponds to the $s-t$ arc cut $\partial(S)$, and the capacity $c(S,T)$ is just the number of edges in the $s-t$ arc cut. From Theorem 28.5 and Theorem 28.8, we know that there is an integer maximum flow with value $c(S,T)$.

The arcs all have capacity 1, so an integer maximum flow has $f(x,y) \in \{0,1\}$ for each arc $(x,y) \in E(D)$. Define a subgraph D' of D by keeping only those arcs of D with flow 1 along them. In this subgraph, the outdegree $\deg^+(s)$ is $c(S,T)$, the value of the flow, and every vertex x other than s and t satisfies a discrete version of flow conservation: the indegree $\deg^-(x)$ is equal to the outdegree $\deg^+(x)$.

Now, to prove Menger's theorem, we will find a set of $c(S,T)$ edge-disjoint $s-t$ paths in the subgraph D'. Now that we've eliminated the arcs we don't need, these paths can be found by wandering aimlessly around D'! Start at s, and follow arbitrary arcs that we have not yet used until, eventually, we get to t.

Question:	Why can't we get stuck at an intermediate vertex, unable to leave without using an arc we've already used?
Answer:	If we're visiting a vertex x for the k^{th} time, we've entered it k times, so $\deg^+(x)$ and $\deg^-(x)$ are at least k. But we've only left it $k-1$ times, so there's an arc out of x we have not used yet.

Once we've arrived at t, start again from s, avoiding all the arcs used before (and in particular, leaving s by a different arc). Repeat until we've left s a total of $\deg^+(s) = c(S,T)$ times, finding a total of $c(S,T)$ edge-disjoint $s-t$ walks. By skipping unnecessary cycles, in the manner of Theorem 3.1, we turn these walks into $s-t$ paths, which remain edge-disjoint to each other. This proves Menger's theorem.

The vertex version of Menger's theorem is also possible to obtain from the maximum flow problem, but it requires some trickery; I will leave the details to you in a practice problem. One useful trick is to give some arcs capacity ∞ when you really don't want them to be part of any network cut. (If you object that ∞ is not an integer, you can also use a large integer constant C which is greater than the capacity of every network cut using only "approved" arcs.)

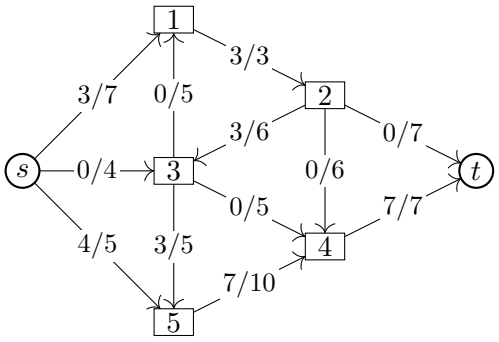
From here, we can go back and obtain results about matchings. From Lemma 27.2, we already know that Kőnig's theorem (Theorem 14.2) can be obtained from Menger's theorem. By

constructing the right network, we can also prove Kőnig's theorem and Hall's theorem (Theorem 15.1) directly.

Why bother? Well, the Ford–Fulkerson algorithm, and later more efficient variants, are computationally useful. Simply determining the capacity of a network cut in the appropriate network (which the Ford–Fulkerson algorithm certainly does) is enough to compute the $s-t$ vertex and edge connectivity in a directed graph. We can use this, in turn, to compute $\kappa(G)$ and $\kappa'(G)$ for an arbitrary graph G, directed or undirected, and this is one of the best ways we have of accomplishing such a thing!

28.7 Practice problems

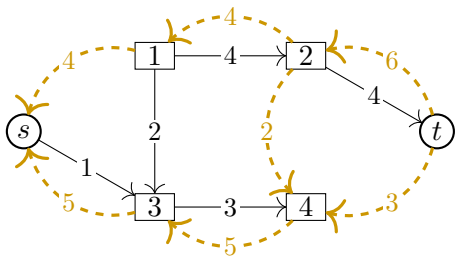
1. Find all possible f-augmenting paths for the flow f given in the diagram below. Determine which one can be used to augment f by the largest amount possible.



2. Find an example of a network in which all arcs have capacity 1 or capacity C, where C can be set to any large value, and in which the Ford–Fulkerson algorithm, by choosing the f-augmenting path badly at each iteration, can take more than C iterations to find a maximum flow.

   (The example given by Edmonds and Karp in [25] has only 4 vertices.)

3. The diagram below gives a residual network for a network flow problem.



   a) Use the residual network to reconstruct the original network, determining the arcs it has and their capacities.

   b) Find the feasible flow which produces this residual network.

   c) Find a network cut whose capacity is equal to the value of the flow.

4. a) Give an example showing that even if the capacities in a network are all integers, it's possible to have a maximum flow f such that not every value $f(x, y)$ is an integer. (All we've proved in this chapter is that the Ford–Fulkerson algorithm will never find such a maximum flow.)

   b) Prove that if f and g are two feasible flows, then the flow $h = \frac{f+g}{2}$ (that is, the flow h with $h(x, y) = \frac{f(x,y)+g(x,y)}{2}$ for all arcs (x, y)) is also a feasible flow, and if f and g are both maximum flows, then so is g.

5. To prove the vertex version of Menger's theorem using maximum flows, we have to get a bit creative. In this practice problem, that's your job!

   Given a directed graph D with two vertices s and t, not adjacent to each other, construct a network N with integer capacities and the following properties:

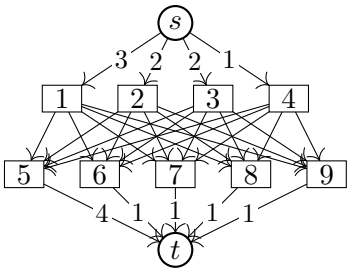
   - Given a k-vertex $s - t$ cut in D, we can find a network cut in N with capacity k.
   - Given an integer flow in N with value k, we can find a set of k internally-disjoint $s - t$ paths in D.

   (If D has n vertices and m arcs, then it's possible to construct N so that it has $2n - 2$ vertices and $m + n - 2$ arcs.)

6. Prove that by being very clever about our choice of paths, it is possible to find any maximum flow at the "greedy" stage of the Ford–Fulkerson algorithm, without needing to use the residual network or any backward arcs.

   (This fact is not very useful in practice, since knowing the right paths to use requires knowing the desired maximum flow in advance.)

7. a) Find an integer flow in the network below. (The arcs whose capacities are not marked should all be assumed to have capacity 1.)



   b) Find a bipartite graph with bipartition $(A, B) = (\{1, 2, 3, 4\}, \{5, 6, 7, 8, 9\})$ and degree sequence

$$(\deg(1), \deg(2), \ldots, \deg(9)) = (3, 2, 2, 1, 4, 1, 1, 1, 1).$$

   c) Explain the connection between parts (a) and (b), and how the network could be modified to find a bipartite graph with a different bipartition and degree sequence.

8. In Chapter 17, we defined a 2-*factor* in a graph G to be a 2-regular spanning subgraph of G.

   Explain how to use the Ford–Fulkerson algorithm to solve the following problem: given a graph G, either find a 2-factor in G, or determine that no 2-factor exists.

Appendix

A A review of proof-writing

The purpose of this chapter

While I hope that these little sections at the beginning of each chapter are always useful, the note in this chapter is much more important to read!

It would be far too ambitious of me to hope to write a single chapter that will teach you how to write a proof. Entire books have been written on this subject. Here are a couple that have been made freely available online by their authors:

- *Book of Proof* by Richard Hammack [48].

  You can find it online at https://richardhammack.github.io/BookOfProof/.

- *An Infinite Descent into Pure Mathematics* by Clive Newstead [77].

  You can find it online at https://infinitedescent.xyz/.

A classic book that should also be mentioned is *How to Solve It* by George Pólya [82]. It is not a book about the method of writing a proof, but will give you advice on how to come up with an idea for it, which is maybe even more valuable.

Usually, when you learn proof-writing, you begin by proving statements that are mathematically easy, so you don't have to struggle with two things at once. After you've done that, you might not yet be comfortable with writing proofs on topics that challenge you mathematically. If this is the level you're at, then this chapter is for you.

I will try to prepare you for the kind of proofs you will encounter in this book by showing you how the general ideas of proof-writing specialize to graph theory. A proof is a proof in every area of math, but there are some ideas that show up in graph theory much more often than in number theory or real analysis, for example. When you get used to proofs in several areas of math, you will be an experienced mathematician. You'll be able to jump into a new topic much more confidently.

So that you can benefit from this chapter early, I'll limit myself to examples from the first part of this book (Chapters 1 through 4), but I will mention chapters later in the book where you'll see more examples.

A.1 Conditional statements

In the moment when you first start trying to prove a theorem, you may not yet know how the proof will go. However, just from the statement of the theorem, you can have a rough overall idea of the strategy you need to take. This is a good way to fight "blank page syndrome", where you're staring at a problem and have no clue what to write!

So what do we look for in a statement? A big part of it is conditional statements: statements that can be expressed in the form "If P, then Q". If you're trying to prove such a statement, you can try:

- A **direct proof**, where you assume that P is true, and try to prove that Q is true.
- A **proof by contrapositive**, where you assume the negation of Q, and try to prove the negation of P. (This is a direct proof of the **contrapositive**, "If not Q, then not P", which is an equivalent form of the original conditional statement.)
- A **proof by contradiction**, where you simultaneously assume both P and the negation of Q, and try to prove any statement you know is false.

A big reason why conditional statements are confusing to beginners is that the informal meaning of "if P, then Q" in the English language is under-specified. If you say a sentence like this in casual conversation, only context can determine what you mean in the case that P is not true. Consider the following examples:

- "If you've lived in Paris your whole life (P), you're French (Q)." Here, Q can still be true even if P is false: many French people have never been to Paris.
- "If you spend at least \$50 (P), you will get free shipping (Q)." Here, the offer implies that you will not get free shipping unless you spend at least \$50: when P is false, Q is also false.
- "If you're hungry (P), there's pizza in the fridge (Q)." Here, the condition is irrelevant, and Q is true regardless of the status of P.

In formal mathematics, an "if P, then Q" statement always goes hand-in-hand with an implicit "and if not P, then anything could be true". It is an assertion that Q is true, but limited only to situations where P is true, without making any claim about situations where P is false. When we want to talk about both situations, and say that when P is false, Q must also be false, the standard formulation is "Q if and only if P".[38]

Even mathematicians are not perfect at maintaining this distinction in all circumstances. One place where you will often see the word "if" misused, from a formal point of view, is in definitions. Though I have avoided it in this book, it is common to see definitions phrased as, for example, "A walk $(x_0, x_1, \ldots, x_l)$ is closed if $x_0 = x_l$." All definitions should be read as if-and-only-if statements: a walk is closed if $x_0 = x_l$, and not otherwise.

Finally, let me say a bit about necessary conditions and sufficient conditions. This is another view of conditional statements. If the statement "If P, then Q" always holds, we say that P is a **sufficient condition** for Q: knowing that P is true is enough to know that Q is true. We call Q a **necessary condition** for P, in the sense that without Q happening, P can't happen

[38] Thus, a mathematician would write, "You will get free shipping if and only if you spend at least \$50."

either: this is a paraphrase of the contrapositive, "If not Q, then not P". We don't always use these terms, though: we use them to emphasize specific ways of thinking about a conditional statement.

We say that P is a sufficient condition for Q to emphasize that we're in a scenario where Q is normally a difficult statement to evaluate, but P is an easy-to-check test that can sometimes tell us that Q is true. For example, Corollary 4.7 is a conditional statement where P is "A graph G with n vertices has at least n edges" and Q is "G contains a cycle". Often, counting vertices and edges is much easier than investigating the graph's structure to see if it has a cycle or not. If we count the vertices and edges and the hypothesis of Corollary 4.7 holds, then we can skip that investigation! The condition we've checked is sufficient to know that the graph has a cycle without looking for it.

We say that Q is a necessary condition for P in the reverse scenario: when P is difficult to investigate, and Q is a simple test. However, with necessary conditions, the test has a different meaning: we learn nothing if we check Q and it is true, but if we check Q and it is false, we learn that P must also be false. For example, Proposition 2.2 tells us that if G and H are isomorphic, then $|V(G)| = |V(H)|$ and $|E(G)| = |E(H)|$. Determining whether two graphs are isomorphic is hard, but counting the vertices and edges is a simple initial test we can do. If $|V(G)| \neq |V(H)|$, or if $|E(G)| \neq |E(H)|$, we can skip the hard work: the necessary condition doesn't hold, so G and H are definitely not isomorphic!

If we have an if-and-only-if relationship between P and Q, then either statement is said to be a **necessary and sufficient condition** for the other. If we discover such a relationship, and one of the statements is a simple test, then we can consider the other statement to be easy to check as well. Knowing whether P is true or false tells us all about Q, and vice versa.

A.2 Quantifiers

Whichever proof strategy you select, it will give you some goals to work toward, and some initial assumptions to work with. Usually, those goals and assumptions will contain quantifiers, which give you further structure. Quantifiers come in two types: existential and universal.

An **existential quantifier** is fancy terminology for a piece of a statement which says that some kind of object exists. The notation "$\exists x \in S, P(x)$" is shorthand for saying, "There exists an object x in the set S for which $P(x)$ is true."

Typically, we prove such a claim by constructing an example; in simple cases, that just means writing down what it is. For example, suppose you want to prove the following statement: "The circulant graph $\mathrm{Ci}_8(3)$ is isomorphic to C_8." Two graphs G and H are isomorphic if there exists an isomorphism between them: a function $\varphi\colon V(G) \to V(H)$ with certain properties. Your proof might begin by defining a function φ, and then checking that it has the properties that make it an isomorphism between $\mathrm{Ci}_8(3)$ and C_8.

I should also describe what happens when an existential statement is part of your assumptions. In that case, you get to summon up an object of the type being described, and start using it in your proof; this can be incredibly helpful! For example, suppose you are giving a direct proof of the following statement: "If a graph G has an $x - y$ walk, then it has an $x - y$ path." The

existence of an $x - y$ walk is one of your assumptions: you get to assume that such a walk exists.

It's often a good idea to describe the object in detail, giving names to its moving parts; you might begin by saying, "Let the sequence $(x_0, x_1, \ldots, x_l)$ be an $x - y$ walk, with $x_0 = x$ and $x_l = y$." Later on in the proof, you will get to manipulate the vertices $x_0, x_1, \ldots, x_l$, and use facts about them that are based on the definition of an $x - y$ walk.

Moving on, a **universal quantifier** says that a statement is always true: it is a universal rule. The notation "$\forall x \in S, P(x)$" is shorthand for saying, "For all objects x in the set S, the statement $P(x)$ is true."

Question:	What if the set S is the empty set?
Answer:	In that case, a statement about all elements of S is automatically true, because there's nothing to check: we say it is **vacuously true**. We don't generally make such statements on purpose, but we might get them as a special case of a more general theorem.

To prove such a statement, we need an argument that applies to every element of S at once. There is a standard way to phrase such an argument: it is to choose an arbitrary element $x \in S$, assuming nothing else about it. Then, your goal is to prove that $P(x)$ is true. This can be easy or hard, but look on the bright side: in this scenario, you both get to start with an element $x \in S$ to work with, and get an idea of how you want your proof to end!

For example, suppose you want to prove the following statement: "For all integers $n \geq 3$, the cycle graph C_n is connected." You would begin by taking an integer $n \geq 3$, and writing the rest of your proof for that value of n. As long as you don't accidentally sneak in any further assumptions about n, your argument will apply to every integer you could have chosen.

The case I find most frustrating is the case of assuming a "for all" statement, because you can't do anything with it when you begin. For example, suppose you have a graph G for which you've made the assumption, "All cycles in G have an even length." (See Chapter 13 to learn more about such graphs.) You can't do anything right away—it's only once you encounter a cycle in G that this assumption "triggers" and tells you that the length of the cycle is even. It's possible that G has no cycles, in which case you will never get to use the assumption.

To give you an overview of the situation at a glance, Figure A.1 describes what happens to both types of quantifiers in both cases: when you assume them, and when you prove them. Since some cases are easier to deal with than others, you can try to change which case you have to deal with by changing your approach.

- You could try a different proof method that changes what you have to do: for example, instead of proving a statement, you might assume its negation.
- You could use a theorem which gives an alternate characterization of an object, with a different type of quantifier.

	When you get to assume it:	**When you have to prove it:**
$\forall x \in S, P(x)$	Nothing can be done right away. Whenever you encounter elements of S, then you can assume that P is true for those elements.	Tells you how to begin and end. Begin by defining an element $x \in S$; you can't control anything else about x. Your goal is to prove $P(x)$.
$\exists x \in S, P(x)$	Tells you how to begin the proof. Begin by defining an element $x \in S$ and assuming that $P(x)$ is true; you can't control anything else about x.	Tells you how to end the proof. You must somehow construct an element $x \in S$ (you don't get to start with one) for which $P(x)$ is true.

Figure A.1: How to use quantifiers in a proof

In more complicated statements, you will have to juggle both kinds of quantifiers at once. An especially important combination to look at is the combination $\forall x, \exists y$: "For every x, there exists a y." This is a very demanding statement to prove, because you must present an example of y, which might have to depend on x; however, you can't make any assumptions about x. Let me give you some examples to compare:

- "The circulant graph $\mathrm{Ci}_8(3)$ is isomorphic to C_8" is a concrete, purely existential statement: no universal quantifiers at all. You can prove that an isomorphism exists by writing down what it is and checking the conditions.
- "For all odd $n \geq 3$, the circulant graph $\mathrm{Ci}_n(2)$ is isomorphic to C_n" has a universal quantifier on n, but an existential quantifier hidden inside the word "isomorphic". You might still be able to write down an isomorphism, but you will have to write it down as a formula in terms of n.
- "Every connected n-vertex graph G in which all vertices have degree 2 is isomorphic to C_n" is even trickier: it has a universal quantifier on a graph G, which is a much more complicated object than a number!

  You will not be able to write down an isomorphism, not even with the aid of formulas, because you can't plug a graph G into a formula. In your proof, you might describe a general strategy for finding an isomorphism, and check that it always works.

This situation is part of the reason why algorithms play a big role in graph theory: a common way to prove that something exists is to give an algorithm for finding it. Theorem 8.3 and Theorem 8.4 are good examples of this technique early in the book.

I should also warn you that often, universal quantifiers are omitted in the statement of a theorem. If the statement of a theorem contains variables that don't have a quantifier attached, this usually means that the theorem should be true for all possible values of those variables; the scope should hopefully be clear from context.

I try to avoid doing this too much, but consider for example Corollary 4.7, which I guess I shouldn't go back and edit because then I won't be able to use it as an example here. The full statement of the corollary is: "If G has n vertices and at least n edges, then G contains a

cycle." Neither G nor n is quantified, so we interpret both of them as having hidden universal quantifiers on them. The statement should be true for all graphs G and for all integers $n \geq 1$.

Question:	How do we know what kind of variables G and n are?
Answer:	Part of this is convention: G usually refers to a graph and n usually refers to an integer. Part of this is context: G is mentioned as having vertices and edges, so it should be a graph, and n is the number of vertices in G, so it can only be a positive integer.

A.3 Unpacking definitions

Quantifiers, implications, and many other parts of a statement can be tucked away inside a definition where you can't easily see them. For example, it is not obvious from reading "G and H are isomorphic" that it's existential ($\exists$) statement: that there exists a function $\varphi\colon V(G) \to V(H)$ which is an isomorphism. (Inside the definition of an isomorphism, more quantifiers are tucked away!)

Question:	What are the quantifiers in "G is connected?"
Answer:	It's a $\forall\exists$ statement: for every two vertices $x, y \in V(G)$, there exists an $x - y$ walk.

Question:	What are the quantifiers in "The maximum degree of G is k?"
Answer:	It has two parts: a $\forall$ statement that every vertex has degree at most k, and an $\exists$ statement that there exists a vertex of degree k. (More on such statements later!)

This means that there's an important first step you have to keep in mind whenever you write a proof: you must unpack all the definitions you're working with to get at the moving parts inside them!

Let's look at an example of this: the proof of Lemma 3.3. This lemma claims that the relation $\leftrightsquigarrow$ on the vertices of a graph G is an equivalence relation, where $x \leftrightsquigarrow y$ is defined to mean that there is an $x - y$ walk in G.

To begin with, we are proving that something is an equivalence relation. By definition, an equivalence relation is a relation which is reflexive, symmetric, and transitive. So right away, we know that our proof will have three parts, one where we prove each part of the definition.

Let's look at just one of these: proving that $\leftrightsquigarrow$ is symmetric. The definition of this is that for all x and y (which, in this case, are vertices of G), if $x \leftrightsquigarrow y$, then $y \leftrightsquigarrow x$.

Question:	Assuming that we choose to write a direct proof (which there's no reason not to do), which quadrant of Figure A.1 are we in?
Answer:	The top right quadrant: we are proving a universal statement about all pairs of vertices $x, y \in V(G)$.

Therefore we begin by choosing arbitrary vertices $x, y \in V(G)$. We assume that $x \leftrightsquigarrow y$ is true, and set ourselves a goal: to prove that $y \leftrightsquigarrow x$ is true. At this point, we must also unpack the definition of $x \leftrightsquigarrow y$. We are assuming that there is an $x - y$ walk in G; we are proving that there is a $y - x$ walk in G.

Question:	Which quadrants of Figure A.1 do these fall under?
Answer:	The bottom two quadrants: both the assumption we make about x and y and the property we want to prove are existential statements. We get to summon an $x - y$ walk out of nowhere, but we will have to construct a $y - x$ walk.

It is not very helpful to use the existence assumption merely by saying something like, "Let W be an $x - y$ walk in G." To get anything useful out of the assumption, we should unpack the definition of an $x - y$ walk. We say, "Let the sequence $(x_0, x_1, \ldots, x_l)$ be an $x - y$ walk, with $x_0 = x$ and $x_l = y$."

Looking ahead, we see that we'll want to define a new sequence of vertices, which we will then prove is a $y - x$ walk. It is only at this point that we get to the problem-solving step of the proof! To figure out how to get a $y - x$ walk out of an $x - y$ walk, we might look at some small examples to get a feeling for the problem. With experience comes an "intuition" for things to try, which is just a way to say that you've seen similar problems before and have some guesses about what might work.

To summarize, here is the "scaffolding" of a proof that $\leftrightsquigarrow$ is symmetric; only the sections with "..." are left for us to figure out. (Not every proof is as heavy in definitions, and so you won't always have as much unpacking to do.)

Proof. Let x and y be two arbitrary vertices, and suppose that $x \leftrightsquigarrow y$: that there is a $x - y$ walk in G. Our goal is to prove that there is also a $y - x$ walk in G.

Let $(x_0, x_1, \ldots, x_l)$ be an $x - y$ walk (with $x_0 = x$ and $x_l = y$). Then define a new sequence of vertices by ...

This sequence of vertices starts at y, ends at x, and consecutive vertices in the sequence are adjacent because ..., proving that it is a $y - x$ walk.

Since a $y - x$ walk exists, we conclude that $y \leftrightsquigarrow x$. Since this is true for all pairs of vertices $x, y \in V(G)$, we conclude that that $\leftrightsquigarrow$ is symmetric. □

I should also mention that there is a danger to unpacking too far. If you unpack every definition you can, you are forced to write a "low-level" proof that works with the fundamental notions of graph theory. The alternative is a "high-level" proof, where you stop early to apply a theorem you know to an object without unpacking it.

For example, in Chapter 4, we gave Theorem 4.4 a low-level proof: we proved that a cycle exists by listing out the vertices of a walk representing it. Later, we gave Corollary 4.7 a more high-level proof: we proved that a cycle exists by applying Theorem 4.4, without interacting directly with the definition of a cycle.

A.4 Optimization problems

Many definitions in graph theory involve solving an optimization problem: they are phrased in terms of the biggest or smallest thing of a certain type. In the first part of the textbook, this includes:

- The distance between two vertices x and y: the length of the shortest $x - y$ walk.
- The minimum and maximum degree of a graph G: the smallest and largest, respectively, of the degrees of the vertices of G.

When we say, "The distance between vertices x and y is 5," we are making two separate claims. First, we are saying that there is in fact an $x - y$ walk of length 5. Second, we are saying that this is the shortest walk: all $x - y$ walks have length at least 5. The first claim is an existential ($\exists$) statement, and the second claim is a universal ($\forall$) statement.

Question:	What statement about the distance $d(x, y)$ do we make if we only say that there is an $x - y$ walk of length 5?
Answer:	This is equivalent to saying that $d(x, y) \leq 5$. The distance can't be longer than 5, because we've already found walk of length 5, but it might be shorter.

Question:	What statement about the distance $d(x, y)$ do we make if we only say that that all $x - y$ walks have length at least 5?
Answer:	This is equivalent to saying that $d(x, y) \geq 5$. We can't do better than 5, but we don't know if we can actually achieve 5.

What we see for proofs about the distance between two vertices is true in general. An upper bound on a minimization problem is an existential statement: we can prove it by giving a single example of a solution that achieves that bound. A lower bound on a minimization problem is a universal statement: we can prove it by proving that no solution can do better. For maximization problems, the situation is reversed.

Let's look at an example.

Proposition A.1. *Let G be the graph with vertex set $V(G) = \{1, 2, \ldots, 100\}$ in which two vertices x and y are adjacent if and only if $|x - y|$ is the square of a positive integer. (For example, vertices 7 and 71 are adjacent, because $|7 - 71| = 64 = 8^2$.)*

Then G has maximum degree $\Delta(G) = 14$.

Proof. First, we prove that a vertex of degree 14 exists, by example. The vertex 50 is adjacent to the 14 vertices

$$1, 14, 25, 34, 41, 46, 49, 51, 54, 59, 66, 75, 86, 99.$$

We don't have to check them all by hand: these are the vertices $50 - k^2$ for $k = 1, 2, \ldots, 7$ and $50 + k^2$ for $k = 1, 2, \ldots, 7$. All we have to do is to check that the extreme values $50 - 7^2 = 1$ and $50 + 7^2 = 99$ still fall within $V(G)$.

This example proves that $\Delta(G) \geq 14$; to prove that $\Delta(G) \leq 14$, we prove that all vertices have degree at most 14.

Let x be an arbitrary vertex. From the example we looked at, we can see that to find the degree of x, we need to count how many values of the form $x - k^2$ or $x + k^2$ fall with in the range $\{1, 2, \ldots, 100\}$. Let a be the largest integer such that $x - a^2 \geq 1$, and let b be the largest integer such that $x + b^2 \leq 100$; then the neighbors of x are $x - 1^2, x - 2^2, \ldots, x - a^2$ and $x + 1^2, x + 2^2, \ldots, x + b^2$, and x has degree $a + b$.

Question:	What is the largest possible value of a?
Answer:	It is 9: even if x is as large as possible, that is if $x = 100$, then $x - 10^2$ is still not in the range $\{1, 2, \ldots, 100\}$.

Similarly, we can prove that the largest possible value of b is 9, giving us an upper bound of $9 + 9 = 18$ on the degree of x, and on the maximum degree. But that's not good enough; we wanted $\Delta(G) \leq 14$, not $\Delta(G) \leq 18$.

Question:	What prevents us from making both a and b this large simultaneously?
Answer:	To get a to reach 9, we need x to be close to the end of the range. To get b to reach 9, we need x to be close to the start of the range.

We know that having $a = 7$ and $b = 7$ are possible when $x = 50$, so let's rule out the remaining possibilities. If $a \in \{8, 9\}$, then $x - 8^2 \geq 1$, so x is at least $1 + 8^2 = 65$; but now, since $65 + 6^2 = 101$ is too big, we learn that $b \leq 5$. Similarly, if $b \in \{8, 9\}$, then $x + 8^2 \leq 100$, so x is at most $100 - 8^2 = 36$; but now, since $36 - 6^2 = 0$ is too small, we learn that $a \leq 5$. So in these cases, no better solution than $a + b = 9 + 5 = 14$ or $a + b = 5 + 9 = 14$ is possible.

Question:	Is $(a, b) = (9, 5)$ or $(a, b) = (5, 9)$ actually achievable?
Answer:	No: for example, $a = 9$ requires $x \geq 82$, and $b = 5$ requires $x \leq 75$. This is fine for our proof: we are done with the stage that proves existence, now we are only ruling out options, and we don't care about ruling out options that don't contradict the upper bound $\Delta(G) \leq 14$ we want.

When $a \in \{8, 9\}$ or $b \in \{8, 9\}$, we have $\deg(x) = a + b \leq 14$. But if $a \leq 7$ and $b \leq 7$, we also have $\deg(x) = a + b \leq 14$. Therefore $\deg(x) \leq 14$ no matter what x is. This tells us that $\Delta(G) \leq 14$, which completes the overall proof. □

In Chapter 5, the diameter of a graph G is introduced: the largest distance between two vertices of G. The definition of diameter has one optimization problem nested inside another! This means that both lower and upper bounds on the diameter of a graph unpack to statements with multiple quantifiers.

For example, suppose we want to prove that the diameter of G is at most k. Then we want to prove that $d(x, y) \leq k$ for all pairs of vertices x, y, which is a $\forall\exists$-type statement: for all x and y, there exists an $x - y$ walk of length at most k.

Question:	What if we try to prove that the diameter of G is at least 5?
Answer:	This will be an $\exists\forall$-type statement: we want to prove that there exist vertices x and y such that all $x - y$ walks have length at least 5.

Here are some of the notable optimization problems introduced later in the book:

- The maximum matching problem and the minimum vertex cover problem, introduced in Chapter 13.
- The independence number $\alpha(G)$ and clique number $\omega(G)$, defined in Chapter 18.
- The vertex and edge coloring problems studied in Chapter 19 and Chapter 20.
- The connectivity $\kappa(G)$ and several related parameters, defined in Chapter 26.

A.5 Algorithms

Algorithms play a role in graph theory for two reasons.

First of all, there are many applications of graph theory in software, so computer scientists think about graph algorithms.

Second, even pure mathematicians that don't care about computer science at all can be interested in graph algorithms as an aid in proof-writing. One way to prove that an object exists is to give an algorithm for finding it.

In both cases, but especially in the second case, it's important for the algorithm to be accompanied by a proof of correctness. In most respects, this is a proof like any other. There are a few proof techniques that are unusually common when dealing with algorithms, which we'll see in a moment. Before that, there's one feature of the proof that I want to point out, because it's easy to miss if you've never thought about it before. It is important to prove that the algorithm eventually stops, and doesn't just keep iterating forever!

In the first part of the book, there is only one notable algorithm: breadth-first search, which we use to compute distances in a graph at the end of Chapter 3. Breadth-first search continues to be useful in algorithms throughout this book: as late as Chapter 28, we return to it to find

(a) A cycle in the cube graph

(b) Replacing the cycle

Figure A.2: An example of the operation in Problem A.1

a shortest path in a residual network. In this algorithm, the stopping condition is relatively straightforward. In each step, we either explore at least one new vertex, or we stop because we didn't find any new vertices. We cannot keep exploring new vertices forever: eventually, we run out of vertices in the graph to explore!

Such situations are common, but not universal. In this textbook, the most sophisticated proof that an algorithm stops is Theorem 28.7, for an algorithm to find maximum flows. In that case, the algorithm deals with real numbers, and it's possible to keep increasing a real number forever even if there's an upper bound on the number; this is what makes the analysis tricky!

To see a tricky proof that an algorithm stops that only needs topics from the first part of the book, let me give an example from the 2019 Princeton University Mathematics Competition (PUMaC): problem 1 from the individual final round in Division A [86]. The problem does not talk about algorithms, but it describes an iterated procedure performed on a graph, so many of the same ideas come up. Here it is:

Problem A.1. *Given a connected*[39] *graph G and cycle C in it, we can perform the following operation: add another vertex v to the graph, connect it to all vertices in C and erase all the edges from C. Prove that we cannot perform the operation indefinitely on a given graph.*

An example of this operation is shown in Figure A.2. Since the graph gets bigger and bigger every time we perform the operation, it's not at all obvious that we'll eventually have to stop!

To solve the problem, we'll need two lemmas.

Lemma A.2. *After the operation in Problem A.1 is performed on a connected graph, the result is another connected graph.*

Proof. Let G be the graph we started with and let H be the result of performing the operation.

For any two vertices $x, y \in V(G)$, there is an $x - y$ walk in G, because G is connected. We can turn this $x - y$ walk in G into an $x - y$ walk in H, even though not all edges of G are still present in H. Every time that the walk goes from a vertex u to a vertex w along an edge of C, replace that step by two steps, u to v to w.

[39] The problem statement on the PUMaC website does not say that G is connected, but the official solution assumes a connected graph. If G is not connected, the same argument can be applied to each connected component. I've decided to add the assumption just so that we can skip this step.

This proves that all vertices of $V(G)$ are in the same connected component of H. In H, there is one more vertex: the vertex v we added. It is in the same component as the other vertices, because it is adjacent to the vertices on C; this proves that H is connected. □

Lemma A.2 is a good example of proving an **invariant** of an algorithm (not to be confused with a graph invariant). By showing that some property (in this case, the connectedness of the graph) does not change after we perform the operation once, we can conclude that it never changes. In this case, it follows from Lemma A.2 that no matter how many times we perform the operation in Problem A.1, we will have a connected graph.

We can obtain another invariant by counting the vertices and edges.

Question:	In Figure A.2, what is the number of vertices before and after the operation, and what is the number of edges?
Answer:	The number of vertices goes from 8 to 9. The number of edges remains at 12: we deleted 4 edges on the cycle, but added 4 edges from the vertices of the cycle to the new vertex.

This example is probably enough to guess what happens in general:

Lemma A.3. *After the operation in Problem A.1 is performed on a graph with n vertices and m edges, the result is a graph with $n+1$ vertices and m edges.*

Proof. The new graph has $n+1$ vertices because 1 vertex is added and none are removed.

Let the cycle C used in the operation have length k; then C has k vertices and k edges. Deleting the edges of C reduces the number of edges from m to $m-k$, but adding an edge from v to each vertex of C increases the number of edges from $m-k$ back to m. □

Technically, the number of vertices is not an invariant of the algorithm, but a monovariant: rather than always staying the same, it increases in a predictable way.

Armed with these two lemmas, we can explain why the operation can't be continued forever.

Question:	Suppose that we could perform the operation 100 times on the cube graph. What can you say about the result?
Answer:	The result must be a connected graph (by Lemma A.2) with 108 vertices and 12 edges (by Lemma A.3).

Question:	Why is this ridiculous?
Answer:	In a graph with 108 vertices and 12 edges, there are not even enough edges to give every vertex a neighbor! The graph must have many different connected components which are isolated vertices; it certainly cannot be connected.

Generalizing this logic, suppose that we start with a connected graph G that has n vertices and m edges. Assume for the sake of contradiction that we can perform the operation at least $2m$ times. When this is done, we have a connected graph G (by Lemma A.2) with $2m + n$ vertices and m edges (by Lemma A.3).

To see the problem clearly, apply the handshake lemma (Lemma 4.1). If every vertex in G has degree at least 1, then the sum of all $2m + n$ degrees is at least $2m + n$, but the handshake lemma says that the sum of degrees is only $2m$. Therefore G must have some vertices of degree 0. Each such vertex is a connected component all by itself: somehow, G is not connected! This is a contradiction, so it must be impossible to perform the operation $2m$ times.

In fact, using techniques from Chapter 10, we can determine the exact number of times we can perform the operation before we have to stop. When we are forced to stop, we must have a graph that is connected but has no cycles: by Theorem 10.2, this graph must be a tree! A tree with m edges has $m+1$ vertices, so if we started with a connected graph that has m edges and n vertices, we will be able to perform the operation $m - n + 1$ times: no more, and no less.

A.6 The extremal principle

Not all proofs of existence are algorithms that tell you how to construct the object we want. There are many proof techniques that let us prove something exists without giving us any clues about how to find one. (For example, in Theorem 18.5, we prove that a graph with a certain property exists by showing that a randomly chosen graph has a positive probability of having that property.)

A notable proof technique of this type is the extremal principle. In the first part of the book, there are two examples of its use:

- In Theorem 3.1, to prove that an $x - y$ path exists, we chose a shortest $x - y$ walk, and then proved that it represents a path.
- In Theorem 4.4, to prove that a cycle exists, we chose a longest path, and then proved that an initial segment of that path can be turned into a cycle.

In general, the **extremal principle** is a way we can make use of an assumption that something exists (as in the bottom left quadrant of Figure A.1) and strengthen that assumption for free. Instead of taking an arbitrary object of that type, we take an object which is as good as possible by some measure.

There are two caveats to this. First of all, you have to know that there are objects of that type. When we chose a shortest $x - y$ walk in the proof of Theorem 3.1, for example, the existence of $x - y$ walks was one of our assumptions. If it was not, then the proof would not be valid: we can't take a shortest $x - y$ walk if there's a possibility that x and y are not in the same connected component.

Question:	When we chose a longest path in the proof of Theorem 4.4, how did we know that a path exists?
Answer:	If nothing else, a graph with at least one vertex should have a path of length 0. (With the assumption that the graph has minimum degree 2, we can even guarantee slightly longer paths than this!)

Second, it must be guaranteed that there is a best object, however you decide to judge "best". Ties are okay, but when there are infinitely many options to choose from, it's possible that no matter which object you choose, there's a better one. In graph theory, we are usually dealing with non-negative integers. In that case, it's always okay to pick the smallest value (as in the case of a shortest $x - y$ walk). In situations where there's an upper bound on the value, we can also pick the largest; however, that's not valid in general.

Question:	Why is there guaranteed to be a longest path in any graph?
Answer:	In a graph with n vertices, a path can also contain at most n vertices, which means that it can have length at most $n - 1$. (There's not necessarily a path of length $n - 1$; this is just an upper bound.)

When should you consider using the extremal principle? Although you could try using it at any time, I've found that there's a specific kind of situation where it's convenient to use: when you can imagine doing something over and over again, the extremal principle can let you "skip to the end" of that process. That's vague, so let me explain how it works in the two examples of this section.

To prove Theorem 3.1, you can imagine taking an $x - y$ walk and cleaning it up to make it represent an $x - y$ path, step by step. How can we do that? Well, an "un-path-like" thing for a walk to do is to revisit a vertex z. Whenever the walk does that, we can eliminate a visit to z by skipping the segment of the walk between the first visit to z and the last. Every time we do this, it makes the walk shorter. Therefore if we skip ahead to an $x - y$ walk that is as short as possible, it must not be possible to do this again.

To prove Theorem 4.4, you can imagine walking around the graph arbitrarily (but without backtracking along the same edge) until you come back to a vertex. When you've visited a vertex for a second time, the trajectory from your first to your second visit must form a cycle. How can we skip ahead to the end here? Well, every time we don't come back to a vertex, we end up making a longer and longer path. So if we skip ahead to the longest path in the graph, we'll be forced to come back to a vertex in the very next step.

A.7 Practice problems

1. Rewrite each of the following statements to make the quantifiers and logical implications inside it explicit.

   a) Two isomorphic graphs always have the same number of edges.

   b) Every n-vertex graph in which all vertices have degree at least $\frac{n-1}{2}$ is connected.

c) Every graph with more edges than vertices has a vertex of degree at least 3.

d) The difference between minimum and maximum degree in a graph can be arbitrarily large.

2. Find the logical negation of each of the following statements, simplifying as much as possible so that no negations of complicated clauses are left. (It is okay if you are left with elementary negations like "x is not adjacent to y".)

a) Every k-vertex subgraph of G has at most k edges.

b) There is a set of k vertices in G with no edges between them.

c) There is a vertex in G adjacent to all other vertices.

d) For all $n \geq 1$, if an n-vertex graph contains no copies of G, then it can have at most $\frac{1}{3}n^2$ edges.

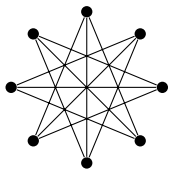
e) For every vertex x of G, there is another vertex y such that every $x - y$ walk in G has length at least 10.

f) G contains two vertices x and y such that for every vertex z other than x or y, if z is adjacent to x, then z is also adjacent to y.

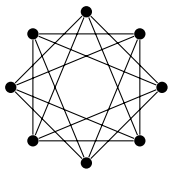
g) Every graph theorist has a friend that knows somebody the graph theorist doesn't know.

h) Every textbook has a practice problem that cannot be solved.

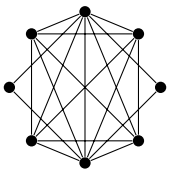
3. Use the three graphs below to answer the questions that follow.



Graph I



Graph II



Graph III

a) If G is a planar graph with $n \geq 3$ vertices, we know that G has at most $3n - 6$ edges. Using this condition and no other properties of planar graphs, what can we say about graphs I, II, and III?

b) If G is a graph with $n \geq 3$ vertices and minimum degree at least $n/2$, we know that G is Hamiltonian. Using this condition and no other properties of Hamiltonian graphs, what can we say about graphs I, II, and III?

c) A connected graph G is Eulerian if and only if every vertex has an even degree. Using this condition and no other properties of Eulerian graphs, what can we say about graphs I, II, and III?

4. a) Prove that the circulant graph $\mathrm{Ci}_8(3)$ is isomorphic to the cycle graph C_8.

b) Prove that for all odd $n \geq 3$, the circulant graph $\mathrm{Ci}_n(2)$ is isomorphic to C_n.

c) Prove that for even $n \geq 4$, the circulant graph $\mathrm{Ci}_n(2)$ is not isomorphic to C_n.

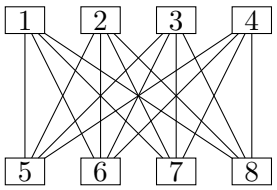
5. Find the error in this proof of the statement "Graphs never have edges".

   > Let G be any graph, and let x be an arbitrary vertex. Let $(x, x_1, x_2, \ldots, x_l)$ be the longest walk beginning at x.
   >
   > If there is an edge xy incident to x, then the walk $(x, y, x, x_1, x_2, \ldots, x_l)$ is a longer walk beginning at x. That's a contradiction, because we assumed that we took the longest such walk. So the assumption that the edge xy exists is incorrect: there are no edges incident to x. In other words, x has degree 0.
   >
   > Since x was an arbitrary vertex, all vertices of x have degree 0: in other words, the graph G has no edges at all!

6. Write down a scaffolding for a direct proof of the claim "If a graph G is isomorphic to its complement $\overline{G}$, then it is connected."

   That is, unpack all the definitions in the claim and identify all the quantifiers and logical implications. Then, write down all the assumptions and initial definitions you should make, as well as the conclusions you should eventually draw. You don't have to fill in the details of how to get from the start to the end.

7. Prove that the graph shown below has diameter 2:



8. Starting from the complete graph K_{10}, you repeatedly perform the following operation: select 4 vertices of the graph, and toggle the presence of all 6 edges between them (removing them if they are absent, and adding them if they are present).

   Can you ever end up removing all edges of the graph?

9. A P_3-free graph is a graph that does not have the path graph P_3 as an induced subgraph. In other words, if you pick 3 vertices inside a P_3-free graph, there will never be exactly 2 edges between them: there could be 0 edges, a single edge, or all 3 edges.

   a) Let G be a P_3-free graph, and let $\sim$ be the adjacency relation in G: $x \sim y$ if and only if $xy \in E(G)$.

      Prove that $\sim$ is an equivalence relation on $V(G)$.

   b) What does this tell us about the structure of G? Describe what a connected component of G can look like.

B Proof by induction

The purpose of this chapter

Induction is a key proof technique for all mathematicians, but it is especially powerful—and especially complicated!—in graph theory. So this chapter has two reasons to exist.

First, it is a review of induction; once again, I am imagining that some of my readers might be studying graph theory shortly after their first introduction to rigorous proofs. If you've mostly studied induction in the context of proving a closed form for a recurrence relation or a summation, then going to more general proofs by induction is a big step. I want to help you along by showing you plenty of examples, and by doing my best to help make the logic of induction make sense.

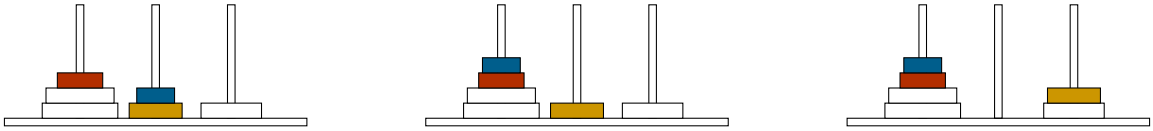
Second, this chapter covers some of the weird things induction can do in graph theory. I especially want to make sure you do not make the mistake in the false Claim B.6, and that you know to use the induction template of "Theorem" B.7 instead, when necessary. I also want to help you discover when induction is a good idea, by describing the kinds of definitions that lend themselves to induction.

These ideas can be applied in other areas of math as well, but because graph theorists study finite objects with not too much structure, it is much more common to be able to tackle a problem by induction.

B.1 The logic of induction

For our first example, let's return to the Towers of Hanoi puzzle, discussed in Chapter 1 and illustrated in Figure B.1. There are three pegs and some number of disks of different sizes stacked on the pegs. In a single move, the top disk on a peg can be moved to the top of a different peg.

The disks stacked on a peg must always form a pyramid, with their sizes increasing from top to bottom. This means that the smallest disk (the one moved from Figure B.1a to Figure B.1b) can always be moved anywhere, but the larger disks are restricted further. From Figure B.1b



(a) A mid-game state (b) The next move (c) Another move

Figure B.1: Two moves in the Towers of Hanoi puzzle

to Figure B.1c, we move a disk from the middle peg to the right peg; it cannot be moved to the lft peg instead, because then it would be on top of a smaller disk.

Initially, the disks are all placed on one peg (forming a pyramid, as they always must). The goal of the puzzle is to move all the disks from one peg to another. Our first task in this chapter will be to prove that this puzzle always has a solution.

Theorem B.1. *No matter how many disks there are, it is possible to solve the Towers of Hanoi puzzle, starting with all the disks stacked in a pyramid on any peg, and ending with all the disks stacked in a pyramid on any other peg.*

If you try to solve this puzzle by moving disks at random, you will not get very far. (I can confirm this experimentally. For the six-disk puzzle in Figure B.1, I performed 1000 computer-simulated attempts to solve the puzzle by randomly chosen valid moves. The median number of moves required to move all six disks onto a peg other than the starting one was 8488.) The reason for this is that it's very rare to see a state where one of the larger disks can be moved.

Question:	If the largest disk in an n-disk Towers of Hanoi puzzle can be moved from the starting peg, how must the disks be arranged?
Answer:	First of all, none of them can be on top of the largest disk. Second, the destination peg to which we want to move the largest disk must also be empty, because the largest disk can't be placed on top of any other. Therefore all $n-1$ other disks must be stacked onto a third peg.

In order to achieve this state, we must move the top $n-1$ disks from one peg to another. In this process, we can ignore the largest disk, which means that we are effectively solving a Towers of Hanoi puzzle with one fewer disk. This makes the problem a perfect setup for an induction proof!

Proof of Theorem B.1. We induct on n, the number of disks. When $n = 1$, we can move the disk from one peg to another in a single step. So the base case holds.

Assume that there is a way to move $n-1$ disks from any peg to any other. Then there is also a solution to the n-disk puzzle, from any starting peg to any ending peg.

1. Using the $(n-1)$-disk solution, move the first $n-1$ disks from the starting peg to the third peg (which is neither the starting nor the ending peg).
2. Move the largest disk from the starting peg to the ending peg.
3. Using the $(n-1)$-disk solution again, move the first $n-1$ disks from the third peg to the ending peg.

By induction, there is a solution for all n. □

This example demonstrates the structure of a proof by induction. Although there are more complicated variants that bend or break some of the rules, an ordinary proof by induction has the following characteristics:

- We must be trying to prove a statement with many cases which can be numbered, usually starting at 0 or 1.

  The first sentence, "We induct on n, the number of disks," is a standard way to explain what the cases are (they are the different versions of the puzzle with different numbers of disks) and how they can be numbered (according to the number of disks, which we are now going to refer to as n).

- We must begin with a **base case**: a self-contained proof of the first case of the statement.

  Here, that's the $n = 1$ case, so at the beginning of the proof, we explain how to solve the 1-disk problem. As in this example, usually the proof of the base case is very short.

- The other part of the proof is an **induction step**: a proof that if any given case of the theorem is true, then the next case is also true.

  As usual, a direct proof of an "if P, then Q" statement begins by assuming P, and trying to prove Q. So we begin the induction step by assuming that the theorem holds in some case, and using this assumption to prove that the theorem also holds in the next case. We refer to this assumption as the **induction hypothesis**.

  The induction step can be written in several slightly different ways. We can assume the n^{th} case of the theorem and prove the $(n+1)^{\text{th}}$, or assume the $(n-1)^{\text{th}}$ case of the theorem and prove the n^{th} case. We can introduce a new variable, proving that if the theorem holds when $n = k$, it also holds when $n = k + 1$ (or that if the theorem holds when $n = k - 1$, it also holds when $n = k$). The choice between these is mostly a matter of taste.

I suspect that many of my readers have already seen proofs by induction, but I am carefully explaining it anyway, because it's a very tricky idea when you first learn it.

It's also often taught without explaining how or why it works. To give you an intuitive understanding, let's prove a few cases of Theorem B.1 without using induction. (In all of these lemmas, by "the puzzle has a solution", we mean that we can move all the disks from any peg to any other peg.)

Lemma B.2. *The Towers of Hanoi puzzle with* 1 *disk has a solution.*

Proof. When no other disks interfere, we can move the disk from any peg to any other in a single step. □

Lemma B.3. *The Towers of Hanoi puzzle with* 2 *disks has a solution.*

Proof. By Lemma B.2, there is a way to move the smallest disk to a different peg. Using this method, move the smallest disk from the starting peg to a third peg: neither the starting nor the ending peg. (In this process, the largest disk won't get in the way, because any other disks can be placed on top of it.)

Now the largest disk is free, and the ending peg is empty, so the largest disk can be moved to the ending peg.

Finally, using the method of Lemma B.2 again, we can move the smallest disk from the peg it's on to the ending peg, which leads to the arrangement we wanted. □

Lemma B.4. *The Towers of Hanoi puzzle with* 3 *disks has a solution.*

Proof. By Lemma B.3, there is a way to move the 2 smallest disks to a different peg. Using this method, move the 2 smallest disks from the starting peg to a third peg: neither the starting nor the ending peg. (In this process, the largest disk won't get in the way, because any other disks can be placed on top of it.)

Now the largest disk is free, and the ending peg is empty, so the largest disk can be moved to the ending peg.

Finally, using the method of Lemma B.2 again, we can move the 2 smallest disks from the peg they're on on to the ending peg, which leads to the arrangement we wanted. □

Lemma B.5. *The Towers of Hanoi puzzle with* 4 *disks has a solution.*

Proof. By Lemma B.4, there is a way to move the 3 smallest disks to a different peg. Using this method, move the 3 smallest disks from the starting peg to a third peg: neither the starting nor the ending peg. (In this process, the largest disk won't get in the way, because any other disks can be placed on top of it.)

Now the largest disk is free, and the ending peg is empty, so the largest disk can be moved to the ending peg.

Finally, using the method of Lemma B.2 again, we can move the 3 smallest disks from the peg they're on on to the ending peg, which leads to the arrangement we wanted. □

Question:	When you compare the proofs of Lemma B.3, Lemma B.4, and Lemma B.5, what do you observe?
Answer:	They are almost identical: they only differ in the number used and in the lemma referenced. (In the case of Lemma B.3, some plural nouns become singular, but that's just a quirk of the English language.)

It is clear that if we wanted to write a proof of a lemma for 5 disks, or 6 disks, or 64 disks, we could do it by copying and pasting and then changing the numbers. So as soon as we have a template which we can copy and paste, we should accept that all those lemmas are true: we know how to obtain a proof of any of them.

This is the logic underlying a proof by induction. The template that we would copy and paste is exactly the induction step, with specific numbers replaced by $n-1$ or n.

Question:	In this copying-and-pasting view of induction, why do we need to prove the base case?
Answer:	Because the proof of Lemma B.2 didn't follow the template: at that point, we didn't have a previous lemma we could cite. So the proof by induction must tell us how to prove the first lemma, as well as how to prove every other lemma.

A common modification to the basic induction template is **strong induction**, where we allow each case of the theorem to rely on multiple smaller cases, and not just the immediately preceding case. For example, if the 6-disk solution involved using a 5-disk solution and a 4-disk solution, or if it unpredictably went back to a k-disk solution for some unknown $k \in \{1, 2, 3, 4, 5\}$, then we'd have to use strong induction.

Question:	Why is this still okay?
Answer:	Because there is no circular reasoning. If we take all the cases the n^{th} case depends on, and then all the cases they depend on, and so on, eventually we end up going back to the base case.

I want to devote most of this chapter to discussing the way induction works in graph theory in particular, but first I want to give one last general piece of advice.

A common tip for getting started solving a problem is to try small cases of the problem first. It is easier to solve specific small examples than to solve a general problem, and by looking at the examples, you can try to make guesses about a general solution. What I want to tell you about is a related idea: if you think you might want to use induction to solve a problem, don't look at the small cases on their own! Look at consecutive cases together, trying to get an idea of what your induction step might be by seeing how one small case can be used to get the next.

So, for example, no matter how you want to solve the Towers of Hanoi puzzle, it's reasonable to look at what happens for 2 or 3 disks to start with. If you want to try a proof by induction, you should look at the 2-disk solution and 3-disk solution side by side, and try to draw connections between the two.

In this case, you might find the 2-disk solution inside the 3-disk solution: once, or even twice. You might also arrive at a useful insight by comparing the lengths of the solutions. Hypothetically, the relationship could have been different: the 3-disk solution could be the 2-disk solution with a short sequence added to the end, or inserted in the middle, or with some transformation applied to each step.

B.2 Induction on graphs

In graph theory, induction is often used in a special way that's slightly different from induction in many other areas of math. Instead of proving that a statement is true for all natural numbers n, we often try to prove it for all graphs, or at least for all graphs with some property relevant to the specific problem.

This tends to work out reasonably well for us, because we have at least two good ways of measuring how big a graph is: we can count the vertices, or we can count the edges. The most common technique is to induct on the number of vertices in a graph. Here, we consider "the statement is true for all graphs with n vertices" to be a single case, and advance through these cases by induction on n. (I will say more later about inducting on the number of edges later; it is less common.)

However, because we're grouping together many graphs into one case, we have to be very careful about how we set up the induction. If we're not, we can make mistakes and prove

false statements, or give incorrect proofs of true statements. The second possibility is harder to catch, because you won't be able to find any counterexamples. To avoid confusing you too much, I will give you an example of the first possibility: a claim that is definitely false.

Claim B.6 (False claim)**.** *For all $n \geq 3$, if G is an n-vertex graph in which every vertex has degree at least 2, then G must have at least $2n - 3$ edges.*

We know this is false, because the cycle graph C_n is a counterexample. Every vertex of C_n has degree 2, but C_n only has n edges, which is less than $2n - 3$ for large n.

Incorrect proof. We induct on n. When $n = 3$, the claim holds, because we need all 3 edges in a 3-vertex graph to exist in order for the assumption to hold, and $3 = 2(3) - 3$.

Assume the claim holds for all $(n - 1)$-vertex graphs: if every in an $(n - 1)$-vertex graph has degree at least 2, then the number of edges is at least $2(n-1)-3$ or $2n-5$. To get an n-vertex graph, we add a vertex; to give it degree at least 2, we need to add at least two new edges. This results in a graph with at least $(2n - 5) + 2$ or at last $2n - 3$ edges.

By induction, the claim is true for all n. □

Just knowing that this proof is incorrect is not enough. We need to understand why it is incorrect, to make sure that we do not do it again.

Question:	The smallest counterexample to Claim B.6 is C_4. Why does the proof fail to consider C_4?
Answer:	When going from 3-vertex graphs to 4-vertex graphs, we try to add a vertex of degree 2 to a 3-vertex graph with minimum degree 2. But C_4 cannot be obtained in this way: if you remove a vertex from it, you're left with P_3, which is not a graph with minimum degree 2.

To avoid this mistake, I have a general theory to propose, and also a practical solution.

The general theory is this. If the possible cases of our theorem were truly the positive integers $n \geq 3$, they would form a chain:

$$3 \to 4 \to 5 \to 6 \to 7 \to 8 \to 9 \to 10 \to \dots$$

If we start at 3 and follow the arrows, we will eventually get to every possibility.

But in Claim B.6, the set of possible cases is really the set of graphs with minimum degree at least 2. Instead of a chain, we can imagine organizing this set into an infinite diagram, a fragment of which is shown in Figure B.2. Here, the arrows represent possible ways to go from a small graph to a bigger one by adding a new vertex of degree 2.

If this is the induction step we intend to use, then a single 3-vertex base case is not enough: by following arrows from that case, we miss many possibilities. We could conceivably write an induction in this way, but we would need many base cases: all the graphs which are not obtained from a smaller graph by following an arrow. At a bare minimum, this includes the cycle graphs, but eventually there are other examples like this. I don't think this is a reasonable

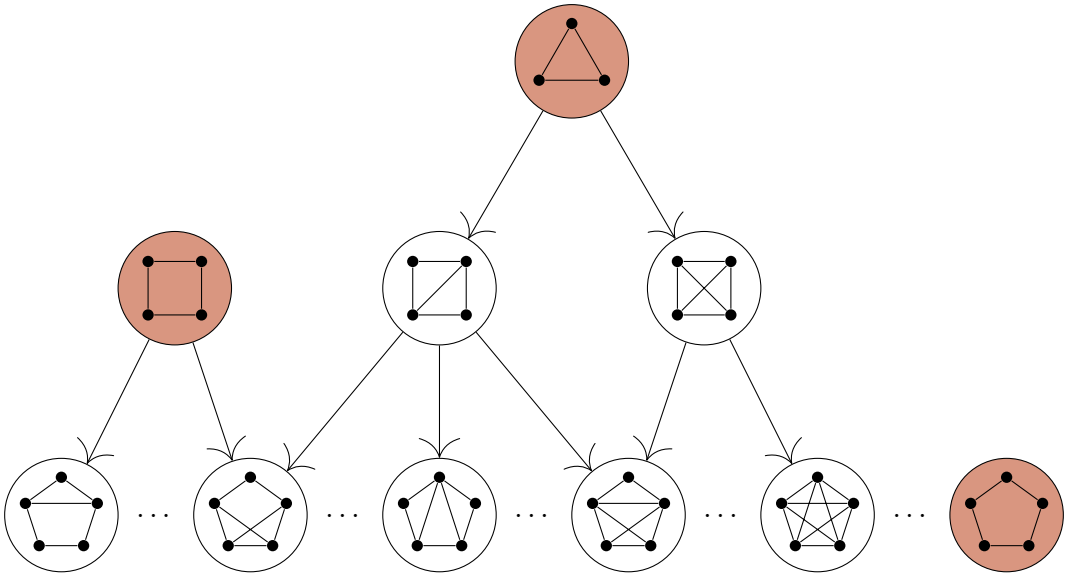


Figure B.2: Organizing the graphs with minimum degree at least 2 according to the induction step of Claim B.6 (not all 5-vertex graphs are included)

way to write an induction proof; I'm just saying that if we wanted to rescue this idea, that's what we'd have to do.

Question:	Is there a 5-vertex graph with minimum degree at least 2, other than C_5, which is not obtained by adding a vertex to a smaller graph of minimum degree at least 2?
Answer:	Yes: we can get another such graph by taking two copies of K_3 that share a vertex of degree 4.

I did, however, promise a practical solution to the problem. This practical solution restricts the way we can write proofs by induction on the vertices of G, allowing only ones that are guaranteed to work. You can think of it as filling in the blanks in the following template:

Theorem B.7 (Induction Template)**.** *If a graph G has property A, it also has property B.*

Proof. We induct on the number of vertices in G. **(Prove a base case here.)**

Now, for some $n > 1$ **(or other lower bound, depending on the base case)**, assume that all $(n-1)$-vertex graphs with property A also have property B. Let G be an n-vertex graph with property A. Our goal is to show that G also has property B.

Let x be a vertex of G **(usually chosen by some clever rule you'll have to come up with)**. Then $G - x$ (the graph obtained from G by deleting x and all edges out of x) also has property A **(by an argument related to the way we chose which vertex to delete)**.

By the induction hypothesis, $G - x$ also has property B. When we add back the vertex x, G also has property B **(by another argument you'll have to come up with)**.

By induction, all graphs with property A also have property B. □

Why does this work, when our previous argument didn't? The key is the step "Let G be an n-vertex graph with property A." We didn't make any assumptions about G. Rather, we started from an arbitrary graph with property A; to apply the induction hypothesis, we cooked up a graph $G - x$ which is an $(n-1)$-vertex graph with property A.

Another way to put it is this: using this induction template prevents the induction from having the kind of structure shown in Figure B.2, by forcing each graph G to have a previous case $G - x$ it is obtained from. Only the graphs covered in the base case are exempt from this.

Using the induction template takes practice to master. (A good example of its use early on in this book is in the proof of Lemma 4.6.) However, it is also helpful, because it is a scaffolding for your proofs by induction: it means you don't have to start from scratch.

Earlier, I mentioned induction on the number of edges of a graph. This is only done a few times in this book:

- When proving the handshake lemma (Lemma 4.1) and later when proving its directed version, Lemma 7.1.
- When proving Theorem 8.3 on cycle decompositions.
- When proving Theorem 10.1 on the number of connected components in a graph with m edges.
- When proving Euler's formula (Theorem 21.4) for plane embeddings.

We can use a variant of the induction template here, deleting an edge xy rather than a vertex x. (The proof of Theorem 8.3 uses strong induction: we delete multiple edges at once.) Repeatedly deleting edges leaves a graph with no edges, but the same number of vertices; all such graphs should be our base cases.

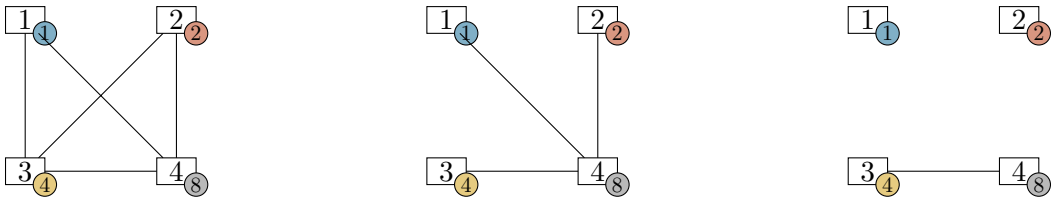
B.3 Inductive definitions

Some structures in graph theory are important because they make it particularly easy to write a proof by induction. There are two that I can point to in this book:

- Trees, which are introduced in Chapter 9. Lemma 10.6 tells us that if T is a tree and x is a degree-1 vertex, then $T - x$ is a tree; moreover, we know from Lemma 10.5 that every tree with multiple vertices has some degree-1 vertices to use Lemma 10.6 with.

  Used in reverse, these two lemmas give us an "inductive definition" of a tree: a tree is either a graph with 1 vertex, or a graph obtained from a smaller tree by adding a new vertex of degree 1.

- 2-connected graphs, which are introduced in Chapter 25. Theorem 25.4 tells us that every 2-connected graph has an ear decomposition, which gives us an inductive definition of a 2-connected graph: it is either a cycle, or a graph obtained from a smaller 2-connected graph by adding an ear.



(a) $t = 4$ (b) $t = 8$ (c) $t = 11$

Figure B.3: 4-vertex threshold graphs with three thresholds t

I don't want to spoil the details of these objects if you haven't read those chapters, even if you're dying to know what it means to add an ear to a graph. But what do I mean by an inductive definition?

Well, in both cases, we are saying something like, "A (type of graph) is either a (base case) or obtained from a smaller (same type of graph) by adding a (small modification)." This is what I mean by an **inductive definition**. You can see how it resembles the structure of a proof by induction—it also makes it easy to write one.

Question:	How can we write a proof by induction using an inductive definition?
Answer:	The base case of the induction is to check that our theorem is true in the base case of the definition. The induction step is to check that if the theorem is true for a graph of this type, it remains true after the small modification.

I want to show you how inductive definitions work in graph theory without getting into either of the advanced examples we learn about elsewhere in the book. So I will give you a different example: a family of graphs called *threshold graphs*. These were defined by Václav Chvátal and Peter Hammer in 1977 to study when multiple inequalities with $\{0, 1\}$-valued variables can be combined into one [18].

Threshold graphs have two definitions. One is the definition they get their name from. A graph G is a threshold graph if there is a function $f\colon V(G) \to \mathbb{R}$ and a value $t \in \mathbb{R}$ such that two vertices x and y are adjacent if and only if $f(x) + f(y) > t$. In other words, adjacency is defined by the sum of vertex labels exceeding a threshold. Figure B.3 shows three examples of 4-vertex threshold graphs: here, the values of f are $0.3, 0.6, 1.2, 2.4$ in all three diagrams, but we require different thresholds t for an edge to exist.

The other definition is the inductive definition. A graph G is a threshold graph if it has only one vertex, or if it can be obtained from a smaller threshold graph in one of two ways:

- adding a new vertex of degree 0, or
- adding a new vertex adjacent to all existing vertices.

Let's practice induction by proving that we get the same class of graphs by either definition!

After I wrote down the proof below for the first time, I realized that I spent half a page just on writing "threshold graph by the ... definition" many times. So to abbreviate, let's call G a

thres graph if it satisfies the inequality definition, and a *hold graph* if it satisfies the inductive definition.

Proposition B.8. *A graph G is a thres graph if and only if it is a hold graph.*

Proof. We will need two proofs by induction, one for each direction of the proof.

First, let's show that if graph G is a hold graph, we can define a function $f\colon V(G) \to \mathbb{R}$ to make it a thres graph. For simplicity, we'll use the threshold $t = 0$.

Our base case here is the 1-vertex graph. We can give the single vertex any value of f we like, and it will be a thres graph, because there isn't a pair of vertices for which to check it.

Suppose now that for some hold graph G, we've defined a function f that also makes it a thres graph. We now need to show that with either of the modifications we do, we can extend f.

- Suppose we add a vertex x to G which is not adjacent to any vertex in $V(G)$, getting a bigger hold graph G'.

  Then to extend f to a function $V(G') \to \mathbb{R}$, we define

  $$f(x) := \min\{-1 - f(y) : y \in V(G)\}.$$

  For each $y \in V(G)$, any value of $f(x)$ equal to or less than $-1 - f(y)$ is enough to make sure that $f(x) + f(y) \leq -1$, preventing an edge xy; we choose $f(x)$ to be the smallest of these values.

- Suppose we add a vertex x to G which is adjacent to every vertex in $V(G)$, getting a bigger hold graph G''.

  Then to extend f to a function $V(G'') \to \mathbb{R}$, we define

  $$f(x) := \max\{1 - f(y) : y \in V(G)\}.$$

  For each $y \in V(G)$, any value of $f(x)$ at least $1 - f(y)$ is enough to make sure that $f(x) + f(y) \geq 1$, ensuring an edge xy: we choose $f(x)$ to be the largest of these values.

In both cases, the larger hold graph continues to be a thres graph with the extended f. We conclude that as we build bigger and bigger hold graphs from the 1-vertex graph, they always remain thres graphs, and therefore by induction, all hold graphs are thres graphs.

That's one part done. Now we show that if G is a thres graph, it is also a hold graph. Since we're trying to establish the inductive definition, we won't be able to use it as a model for induction; instead, we'll use the induction template.

We induct on the number of vertices in G. When G is a 1-vertex thres graph, it is also a hold graph by the base case of the inductive definition.

Now, for some $n > 1$, assume that all $(n-1)$-vertex thres graphs are also hold graphs. Let G be a thres graph, defined using a function $f\colon V(G) \to \mathbb{R}$ and a threshold t.

Question:	For which vertices x is $G - x$ a thres graph?
Answer:	All of them: we can just keep the same function f (with fewer inputs) and the same threshold t.

So we won't be deleting a vertex carefully to make sure we still have a thres graph; we'll be deleting a vertex carefully to make it's easy to check the definition of a hold graph when we put it back.

I propose two options for the deletion. Let x be the vertex of G such that $f(x)$ is as small as possible, and let y be the vertex of G such that $f(y)$ is as large as possible.

Question:	If $f(x) + f(y) \leq t$, what do we know about x?
Answer:	Since not even $f(y)$ is big enough for the sum with $f(x)$ to exceed the threshold, we know $f(x) + f(z) \leq t$ for all vertices z, so x has no neighbors in G.

In this case, consider the thres graph $G - x$. By the induction hypothesis, $G - x$ is also a hold graph. But G is obtained from $G - x$ by adding a new vertex x of degree 0, which means G is also a hold graph.

Question:	If $f(x) + f(y) > t$, what can we do instead?
Answer:	Since even $f(x) + f(y)$ is bigger than t, and $f(x)$ is the smallest value of f there is, we know $f(z) + f(y) > t$ for all vertices z. Therefore y is adjacent to all other vertices of G.

In this case, consider the thres graph $G - y$. By the induction hypothesis, $G - y$ is also a hold graph. But G is obtained from $G - y$ by adding a new vertex y adjacent to all existing vertices, which means G is also a hold graph.

In both cases, we prove G is a hold graph, completing the induction step. By induction, all thres graphs are hold graphs. □

We can compare the incorrect proof of the false Claim B.6 to the way that we used an inductive definition in this proof. Actually, both proofs are written in the same style: we start with a base case and then we grow it. The difference is that in the case of threshold graphs, the inductive definition guarantees that every other threshold graph can be obtained from a 1-vertex graph. (For a demonstration of this, see Figure B.4.) In other words, one base case is all we need!

B.4 Recursive constructions

Let's say that a sequence of graphs $G_1, G_2, G_3, \ldots$ has a **recursive construction** if we have a fixed procedure by which we can build G_n out of G_{n-1} for all $n > 1$. For example, the sequence $Q_1, Q_2, Q_3, \ldots$ of hypercube graphs defined in Chapter 4 has a recursive construction. For all n, we can build Q_n from Q_{n-1} by the following procedure:

1. Start with two disjoint copies of Q_{n-1}.

   If we think of the vertices of Q_{n-1} as $(n-1)$-bit strings, then we can say that the first copy of Q_{n-1} has a vertex $x0$ (x, followed by a 0) for every vertex $x \in V(Q_{n-1})$, and the second copy of Q_{n-1} has a vertex $x1$ for every vertex $x \in V(Q_{n-1})$.

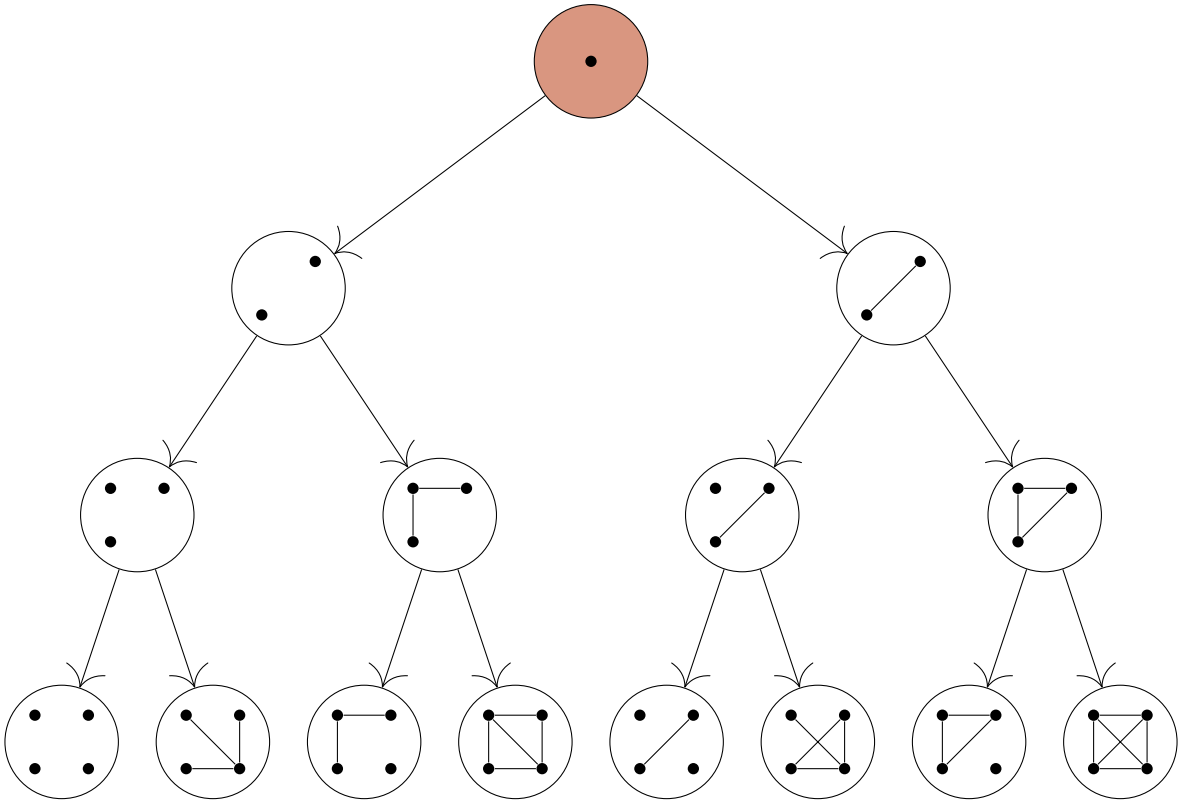


Figure B.4: Building all threshold graphs from a 1-vertex graph

2. In addition to the edges that already exist within each copy of Q_{n-1}, add an edge $\{x0, x1\}$ for all $x \in V(Q_{n-1})$: this is an edge between the two vertices corresponding to x in the two copies of Q_{n-1}.

This is easier to see in a diagram. Figure B.5 shows how we build Q_4 using this recursive construction: the two copies of Q_3 are shown on the left (with a 0 at the end of every vertex) and on the right (with a 1 at the end of every vertex), and the dashed edges are the edges added between the copies.

Other notable examples of recursive constructions in this book include:

- The Mycielski graphs defined in Chapter 19: each graph is the Mycielskian of the previous graph.
- The de Bruijn digraphs defined in Chapter 20: by Proposition 20.3, for all $k \geq 1$, each digraph in the sequence $B(k, 1), B(k, 2), B(k, 3), \ldots$ is the line digraph of the previous graph.
- The iterated barycentric subdivisions used as an example in Chapter 21: to go from each plane embedding to the next, take the barycentric subdivision of every triangular face.

  In general, the word "iterated" in a mathematical term often indicates a recursive construction.

Why do I mention recursive constructions here? You can probably guess why: induction is a good way to prove properties of a sequence of graphs with a recursive construction.

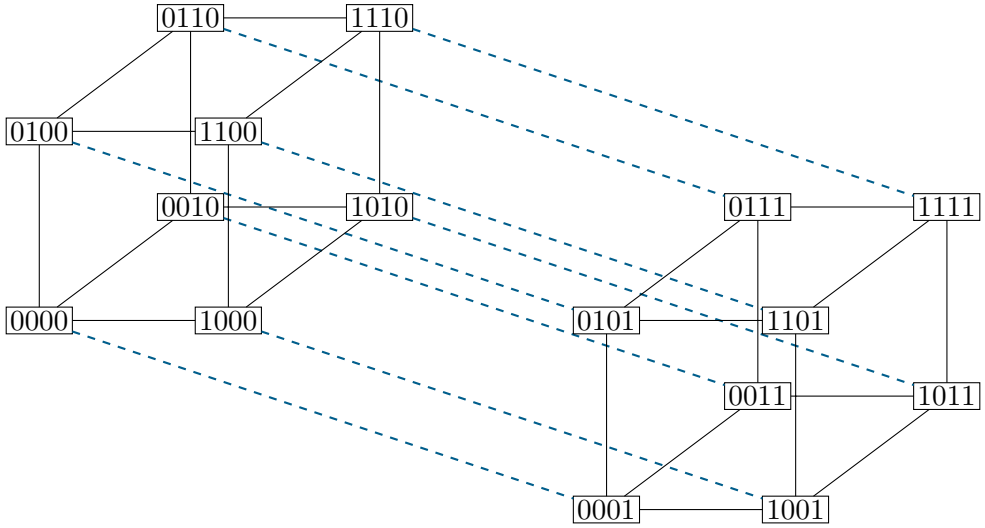


Figure B.5: Using a recursive construction to build Q_4 from two copies of Q_3

Let's start with a simple example, where the proof by induction is actually much more complicated than what we need.

Proposition B.9. *For all $n \geq 1$, the hypercube graph Q_n has 2^n vertices and $n \cdot 2^{n-1}$ edges.*

Proof. We induct on n. When $n = 1$, the hypercube graph Q_1 has two vertices (0 and 1) and one edge between them. Since $2^1 = 1$ and $1 \cdot 2^{1-1} = 1$, both formulas we want to prove are true for $n = 1$.

Now assume, for some $n > 1$, that Q_{n-1} has 2^{n-1} vertices and $(n-1) \cdot 2^{n-2}$ edges. When we construct Q_n recursively, we take two disjoint copies of Q_{n-1}, so we get $2 \cdot 2^{n-1}$ or 2^n vertices. There are $(n-1) \cdot 2^{n-2}$ edges in each copy of Q_{n-1}, and 2^{n-1} edges added between them (one for each vertex of Q_{n-1}), for a total of

$$(n-1) \cdot 2^{n-2} + (n-1) \cdot 2^{n-2} + 2^{n-1} = n \cdot 2^{n-1}$$

edges.[40]

We confirm that if Q_{n-1} has the number of vertices and edges we wanted, then so does Q_n. By induction, the formulas in the proposition hold for all $n \geq 1$. □

Question:	What's the much simpler way to prove that Q_n has $n \cdot 2^{n-1}$ edges?
Answer:	Use the handshake lemma! There are 2^n vertices, and each vertex has degree n, so the sum of degrees is $n \cdot 2^n$, and to get the number of edges, we divide by 2.

[40] It can be tempting to slack off when verifying this identity, since we know what we should get when we simplify if the proof by induction is going to work. Try to resist this temptation and actually do the algebra, or else you risk proving something false one day.

Question:	If we didn't know the formula $n \cdot 2^{n-1}$ ahead of time, could we still use a proof by induction?
Answer:	Yes, but we'd have to solve a recurrence relation to figure out the formula, first. If $f(n)$ is the number of edges, then the argument in our induction step tells us that $f(n) = 2 \cdot f(n-1) + 2^{n-1}$, with $f(1) = 1$.

Here's a quick algebraic way to solve the recurrence relation: first, divide through by 2^n to get $\frac{f(n)}{2^n} = \frac{f(n-1)}{2^{n-1}} + \frac{1}{2}$. Now, each occurrence of $f(k)$ for any k shows up divided by 2^k, so we can define $g(n) = \frac{f(n)}{2^n}$. This has the initial value $g(1) = \frac{f(1)}{2} = \frac{1}{2}$ and a much simpler recurrence relation: $g(n) = g(n-1) + \frac{1}{2}$. If we start with $\frac{1}{2}$ and add $\frac{1}{2}$ each time, then we should get $g(n) = \frac{n}{2}$, and so $f(n) = g(n) \cdot 2^n = n \cdot 2^{n-1}$.

This trick can be used to solve many recurrence relations, but first you have to figure out the right thing to multiply or divide by in order to simplify the recurrence relation.

Here is a second example of induction on Q_n, for which we'll have to work harder.

Proposition B.10. *For all $n \geq 1$, the diameter of Q_n (the largest distance between any two of its vertices) is n.*

Proof. When $n = 1$, Q_n is still the graph with two vertices and 1 edge; this has diameter 1, because the two vertices are at distance 1 from each other.

Now assume, for some $n > 1$, that Q_{n-1} has diameter $n-1$; to prove the induction step, we would like to prove that Q_n has diameter n.

Question:	On the level of walks in a graph, what does our induction hypothesis tell us, and what do we need to prove?
Answer:	We know that for every pair x, y of vertices in Q_{n-1}, there is an $x-y$ walk of length at most $n-1$, and that there is some pair x, y for which no shorter walk exists. We want to show that for every pair x, y of vertices in Q_n, there is an $x-y$ walk of length at most n, and we want to find some pair x, y for which we prove that no shorter walk exists.

First, let's prove that walks of length at most n exist between every pair of vertices in Q_n. If we take two vertices in the same copy of Q_{n-1}, then there is a walk of length at most $n-1$ between them, by the induction hypothesis. So suppose we take two vertices in different copies: without loss of generality, vertices of the form $x0$ and $y1$.

By the induction hypothesis, there is an $x-y$ walk in Q_{n-1} of length at most $n-1$; equivalently, an $x0-y0$ walk in the first copy of Q_{n-1} inside Q_n. By taking that $x0-y0$ walk, and adding the vertex $y1$ to the end (which is adjacent to $y0$), we get an $x0-y1$ walk whose length is only increased by 1: the length is most n, as we wanted.

To find a pair of vertices at distance n, we begin with other half of our induction hypothesis: that Q_{n-1} contains two vertices x, y for which all $x-y$ walks have length at least $n-1$.

Question:	Using x and y, which vertices in Q_n should we pick that we expect to be at distance n?
Answer:	Vertices $x0$ and $y1$, for example: these are as far away as possible and also in different copies of Q_{n-1}.

We can think of a walk in Q_n is a sequence of steps where at each step, we change some coordinate. To get from $x0$ to $y1$, at least once, we need to change the last coordinate, going from $z0$ to $z1$ for some z. Let's just skip all steps where we do that: we will now get a walk from $x0$ to $y0$ in the first copy of Q_{n-1}, and it will be shorter by at least one step.

Question:	What is the motivation for doing this?
Answer:	Our induction hypothesis is an assumption about all $x - y$ walks, so in order to use it, we first need to obtain an $x - y$ walk to apply it to!

This new $x0 - y0$ walk corresponds to an $x - y$ walk in Q_{n-1}, so it has length at least $n - 1$, by our induction hypothesis. The $x0 - y1$ walk has at least one step we skipped, so it has length at least n, completing our lengthy induction step.

We've proved that if Q_{n-1} has diameter $n - 1$, then Q_n has diameter n; by induction, Q_n has diameter n for all $n \geq 1$. □

B.5 Practice problems

1. Let's take a closer look at Theorem B.1.

   a) For $n = 3$ disks, write down the steps of the solution that the proof gives us.

   b) In general, in terms of n, how many steps are there in the solution?

   c) It turns out that the number in part (b) is the least number of steps required to move all the disks from one peg to another. Prove this claim by induction.

2. The sequence $F_0, F_1, F_2, \ldots$ of Fibonacci numbers is defined by $F_0 = 0$, $F_1 = 1$, and the recurrence relation $F_n = F_{n-1} + F_{n-2}$ for all $n \geq 2$.

   A *matching* is defined in Chapter 13 as a spanning subgraph M where every vertex has degree 0 or 1; equivalently, the edges $E(M)$ share no endpoints.

   a) Prove that for all $n \geq 1$, the path graph P_n has F_{n+1} matchings.

   b) Prove that for all $n \geq 3$, the cycle graph C_n has $F_{n+1} + F_{n-1}$ matchings.

3. Prove by induction on n that for all $n \geq 5$, there exists a graph with n vertices and $2n - 4$ edges which has minimum degree 2 and maximum degree 4.

4. Prove that the complement of every threshold graph is also a threshold graph.

5. Let's say that a graph G has "comparable neighborhoods" if, for all $x, y \in V(G)$, either vertex x is adjacent to all the neighbors of y, or vertex y is adjacent to all the neighbors of x. (That is, the *neighborhoods* $N(\{x\})$ and $N(\{y\})$ are *comparable sets*, following definitions in Chapter 15.)

   a) Prove that every threshold graph has comparable neighborhoods.

   b) In the bottom level of Figure B.4, 8 of the 11 possible four-vertex graphs are shown (up to isomorphism). The missing 3 graphs are exactly the 3 graphs that do not have comparable neighborhoods. What are they?

   c) Let G be a graph with comparable neighborhoods. Prove that it either has a vertex adjacent to every other vertex, or a vertex with no neighbors.

   d) Prove by induction that if G has comparable neighborhoods, it is a threshold graph.

6. Let a "quadsum graph" be a graph on at least 4 vertices which is either a 4-vertex cycle or a union $G \cup H$ where G and H are quadsum graphs with 2 vertices and 0 edges in common.

   Prove that all n-vertex quadsum graphs have the same number of edges, and find that number as a function of n.

7. Let a "Hanoi integer" be an integer in which no two adjacent digits are equal (such as 123456, or 271828, but not 144702). We define two operations on Hanoi integers:

   A "tweak" changes the last digit to anything else that's not the same as the previous digit: for example, we may change 123456 to 123450.

   A "twiddle" takes the longest suffix of two digits alternating, as $\ldots xyxy$, and switches the two digits: we may change 123456 to 123465, or 271828 to 271282, or 363636 to 636363. Twiddles are forbidden if they would put a 0 at the beginning of the integer.

   a) Prove that you can change any n-digit Hanoi integer to any other with a combination of $2^n - 1$ tweaks and twiddles.

   b) Prove that $2^n - 1$ operations are required to get from a number written only with the digits 0–4 to a number written only with the digits 5–9.

C License

This book is licensed under the Creative Commons Attribution-ShareAlike 4.0 (CC BY-SA 4.0) license. It is copied below from the Creative Commons website:

https://creativecommons.org/licenses/by-sa/4.0/legalcode

If you have questions or if you would like permission to do something you think the license doesn't let you do, please write me at misha.p.l@gmail.com.

Creative Commons Attribution-ShareAlike 4.0 International Public License

By exercising the Licensed Rights (defined below), You accept and agree to be bound by the terms and conditions of this Creative Commons Attribution-ShareAlike 4.0 International Public License ("Public License"). To the extent this Public License may be interpreted as a contract, You are granted the Licensed Rights in consideration of Your acceptance of these terms and conditions, and the Licensor grants You such rights in consideration of benefits the Licensor receives from making the Licensed Material available under these terms and conditions.

Section 1 – Definitions.

a. Adapted Material means material subject to Copyright and Similar Rights that is derived from or based upon the Licensed Material and in which the Licensed Material is translated, altered, arranged, transformed, or otherwise modified in a manner requiring permission under the Copyright and Similar Rights held by the Licensor. For purposes of this Public License, where the Licensed Material is a musical work, performance, or sound recording, Adapted Material is always produced where the Licensed Material is synched in timed relation with a moving image.

b. Adapter's License means the license You apply to Your Copyright and Similar Rights in Your contributions to Adapted Material in accordance with the terms and conditions of this Public License.

c. BY-SA Compatible License means a license listed at

creativecommons.org/compatiblelicenses,

approved by Creative Commons as essentially the equivalent of this Public License.

d. Copyright and Similar Rights means copyright and/or similar rights closely related to copyright including, without limitation, performance, broadcast, sound recording, and Sui Generis Database Rights, without regard to how the rights are labeled or categorized. For purposes of this Public License, the rights specified in Section 2(b)(1)-(2) are not Copyright and Similar Rights.

e. Effective Technological Measures means those measures that, in the absence of proper authority, may not be circumvented under laws fulfilling obligations under Article 11 of the WIPO Copyright Treaty adopted on December 20, 1996, and/or similar international agreements.

f. Exceptions and Limitations means fair use, fair dealing, and/or any other exception or limitation to Copyright and Similar Rights that applies to Your use of the Licensed Material.

g. License Elements means the license attributes listed in the name of a Creative Commons Public License. The License Elements of this Public License are Attribution and ShareAlike.

h. Licensed Material means the artistic or literary work, database, or other material to which the Licensor applied this Public License.

i. Licensed Rights means the rights granted to You subject to the terms and conditions of this Public License, which are limited to all Copyright and Similar Rights that apply to Your use of the Licensed Material and that the Licensor has authority to license.

j. Licensor means the individual(s) or entity(ies) granting rights under this Public License.

k. Share means to provide material to the public by any means or process that requires permission under the Licensed Rights, such as reproduction, public display, public performance, distribution, dissemination, communication, or importation, and to make material available to the public including in ways that members of the public may access the material from a place and at a time individually chosen by them.

l. Sui Generis Database Rights means rights other than copyright resulting from Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, as amended and/or succeeded, as well as other essentially equivalent rights anywhere in the world.

m. You means the individual or entity exercising the Licensed Rights under this Public License. **Your** has a corresponding meaning.

Section 2 – Scope.

a. **License grant**.

1. Subject to the terms and conditions of this Public License, the Licensor hereby grants You a worldwide, royalty-free, non-sublicensable, non-exclusive, irrevocable license to exercise the Licensed Rights in the Licensed Material to:

A. reproduce and Share the Licensed Material, in whole or in part; and

B. produce, reproduce, and Share Adapted Material.

2. **Exceptions and Limitations**. For the avoidance of doubt, where Exceptions and Limitations apply to Your use, this Public License does not apply, and You do not need to comply with its terms and conditions.

3. **Term**. The term of this Public License is specified in Section 6(a).

4. **Media and formats; technical modifications allowed**. The Licensor authorizes You to exercise the Licensed Rights in all media and formats whether now known or hereafter created, and to make technical modifications necessary to do so. The Licensor waives and/or agrees not to assert any right or authority to forbid You from making technical modifications necessary to exercise the Licensed Rights, including technical modifications necessary to circumvent Effective Technological Measures. For purposes of this Public License, simply making modifications authorized by this Section 2(a)(4) never produces Adapted Material.

5. Downstream recipients.

   A. Offer from the Licensor – Licensed Material. Every recipient of the Licensed Material automatically receives an offer from the Licensor to exercise the Licensed Rights under the terms and conditions of this Public License.

   B. Additional offer from the Licensor – Adapted Material. Every recipient of Adapted Material from You automatically receives an offer from the Licensor to exercise the Licensed Rights in the Adapted Material under the conditions of the Adapter's License You apply.

   C. No downstream restrictions. You may not offer or impose any additional or different terms or conditions on, or apply any Effective Technological Measures to, the Licensed Material if doing so restricts exercise of the Licensed Rights by any recipient of the Licensed Material.

6. No endorsement. Nothing in this Public License constitutes or may be construed as permission to assert or imply that You are, or that Your use of the Licensed Material is, connected with, or sponsored, endorsed, or granted official status by, the Licensor or others designated to receive attribution as provided in Section 3(a)(1)(A)(i).

b. **Other rights**.

1. Moral rights, such as the right of integrity, are not licensed under this Public License, nor are publicity, privacy, and/or other similar personality rights; however, to the extent possible, the Licensor waives and/or agrees not to assert any such rights held by the Licensor to the limited extent necessary to allow You to exercise the Licensed Rights, but not otherwise.

2. Patent and trademark rights are not licensed under this Public License.

3. To the extent possible, the Licensor waives any right to collect royalties from You for the exercise of the Licensed Rights, whether directly or through a collecting society under any voluntary or waivable statutory or compulsory licensing scheme. In all other cases the Licensor expressly reserves any right to collect such royalties.

Section 3 – License Conditions.

Your exercise of the Licensed Rights is expressly made subject to the following conditions.

a. **Attribution**.

1. If You Share the Licensed Material (including in modified form), You must:

   A. retain the following if it is supplied by the Licensor with the Licensed Material:

      i. identification of the creator(s) of the Licensed Material and any others designated to receive attribution, in any reasonable manner requested by the Licensor (including by pseudonym if designated);

      ii. a copyright notice;

      iii. a notice that refers to this Public License;

      iv. a notice that refers to the disclaimer of warranties;

      v. a URI or hyperlink to the Licensed Material to the extent reasonably practicable;

   B. indicate if You modified the Licensed Material and retain an indication of any previous modifications; and

   C. indicate the Licensed Material is licensed under this Public License, and include the text of, or the URI or hyperlink to, this Public License.

2. You may satisfy the conditions in Section 3(a)(1) in any reasonable manner based on the medium, means, and context in which You Share the Licensed Material. For example, it may be reasonable to satisfy the conditions by providing a URI or hyperlink to a resource that includes the required information.

3. If requested by the Licensor, You must remove any of the information required by Section 3(a)(1)(A) to the extent reasonably practicable.

b. **ShareAlike**.

In addition to the conditions in Section 3(a), if You Share Adapted Material You produce, the following conditions also apply.

1. The Adapter's License You apply must be a Creative Commons license with the same License Elements, this version or later, or a BY-SA Compatible License.

2. You must include the text of, or the URI or hyperlink to, the Adapter's License You apply. You may satisfy this condition in any reasonable manner based on the medium, means, and context in which You Share Adapted Material.

3. You may not offer or impose any additional or different terms or conditions on, or apply any Effective Technological Measures to, Adapted Material that restrict exercise of the rights granted under the Adapter's License You apply.

Section 4 – Sui Generis Database Rights.

Where the Licensed Rights include Sui Generis Database Rights that apply to Your use of the Licensed Material:

a. for the avoidance of doubt, Section 2(a)(1) grants You the right to extract, reuse, reproduce, and Share all or a substantial portion of the contents of the database;

b. if You include all or a substantial portion of the database contents in a database in which You have Sui Generis Database Rights, then the database in which You have Sui Generis Database Rights (but not its individual contents) is Adapted Material, including for purposes of Section 3(b); and

c. You must comply with the conditions in Section 3(a) if You Share all or a substantial portion of the contents of the database.

For the avoidance of doubt, this Section 4 supplements and does not replace Your obligations under this Public License where the Licensed Rights include other Copyright and Similar Rights.

Section 5 – Disclaimer of Warranties and Limitation of Liability.

a. Unless otherwise separately undertaken by the Licensor, to the extent possible, the Licensor offers the Licensed Material as-is and as-available, and makes no representations or warranties of any kind concerning the Licensed Material, whether express, implied, statutory, or other. This includes, without limitation, warranties of title, merchantability, fitness for a particular purpose, non-infringement, absence of latent or other defects, accuracy, or the presence or absence of errors, whether or not known or discoverable. Where disclaimers of warranties are not allowed in full or in part, this disclaimer may not apply to You.

b. To the extent possible, in no event will the Licensor be liable to You on any legal theory (including, without limitation, negligence) or otherwise for any direct, special, indirect, incidental, consequential, punitive, exemplary, or other losses, costs, expenses, or damages arising out of this Public License or use of the Licensed Material, even if the Licensor has been advised of the possibility of such losses, costs, expenses, or damages. Where a limitation of liability is not allowed in full or in part, this limitation may not apply to You.

c. The disclaimer of warranties and limitation of liability provided above shall be interpreted in a manner that, to the extent possible, most closely approximates an absolute disclaimer and waiver of all liability.

Section 6 – Term and Termination.

a. This Public License applies for the term of the Copyright and Similar Rights licensed here. However, if You fail to comply with this Public License, then Your rights under this Public License terminate automatically.

b. Where Your right to use the Licensed Material has terminated under Section 6(a), it reinstates:

   1. automatically as of the date the violation is cured, provided it is cured within 30 days of Your discovery of the violation; or

   2. upon express reinstatement by the Licensor.

   For the avoidance of doubt, this Section 6(b) does not affect any right the Licensor may have to seek remedies for Your violations of this Public License.

c. For the avoidance of doubt, the Licensor may also offer the Licensed Material under separate terms or conditions or stop distributing the Licensed Material at any time; however, doing so will not terminate this Public License.

d. Sections 1, 5, 6, 7, and 8 survive termination of this Public License.

Section 7 – Other Terms and Conditions.

a. The Licensor shall not be bound by any additional or different terms or conditions communicated by You unless expressly agreed.

b. Any arrangements, understandings, or agreements regarding the Licensed Material not stated herein are separate from and independent of the terms and conditions of this Public License.

Section 8 – Interpretation.

a. For the avoidance of doubt, this Public License does not, and shall not be interpreted to, reduce, limit, restrict, or impose conditions on any use of the Licensed Material that could lawfully be made without permission under this Public License.

b. To the extent possible, if any provision of this Public License is deemed unenforceable, it shall be automatically reformed to the minimum extent necessary to make it enforceable. If the provision cannot be reformed, it shall be severed from this Public License without affecting the enforceability of the remaining terms and conditions.

c. No term or condition of this Public License will be waived and no failure to comply consented to unless expressly agreed to by the Licensor.

d. Nothing in this Public License constitutes or may be interpreted as a limitation upon, or waiver of, any privileges and immunities that apply to the Licensor or You, including from the legal processes of any jurisdiction or authority.

D Bibliography

[1] Michael Albertson. "You can't paint yourself into a corner". In: *Journal of Combinatorial Theory, Series B* 73 (2 1998), pp. 189–194. DOI: doi.org/10.1006/jctb.1998.1827.

[2] Noga Alon. "A simple algorithm for edge-coloring bipartite multigraphs". In: *Information Processing Letters* 85.6 (2003), pp. 301–302. DOI: 10.1016/S0020-0190(02)00446-5.

[3] Atsushi Amahashi. "On factors with all degrees odd". In: *Graphs and Combinatorics* 1 (1985), pp. 111–114. DOI: 10.1007/BF02582935.

[4] Kenneth Appel and Wolfgang Haken. "Every planar map is four colorable. Part I: Discharging". In: *Illinois Journal of Mathematics* 21 (3 1977), pp. 429–490. DOI: 10.1215/ijm/1256049011.

[5] Kenneth Appel and Wolfgang Haken. "Every planar map is four colorable. Part II: Reducibility". In: *Illinois Journal of Mathematics* 21 (3 1977), pp. 491–567. DOI: 10.1215/ijm/1256049012.

[6] W. W. Rouse Ball. *Mathematical Recreations and Essays*. Fourth edition. New York, NY: Macmillan and Company, 1905. URL: https://www.gutenberg.org/ebooks/26839.

[7] Berkeley Math Circle. *7^{th} Bay Area Mathematical Olympiad*. 2005. URL: https://www.bamo.org/archives/examfiles/bamo2005examsol.pdf (visited on 12/21/2025).

[8] John A. Bondy and Václav. Chvátal. "A method in graph theory". In: *Discrete Mathematics* 15.2 (1976), pp. 111–135. DOI: 10.1016/0012-365X(76)90078-9.

[9] Carl Wilhelm Borchardt. "Über eine Interpolationsformel für eine Art symmetrischer Functionen und über deren Anwendung". In: *Mathematische Abhandlungen der Königlichen Akademie der Wissenschaften zu Berlin* (1860), pp. 1–20. URL: https://archive.org/details/abhandlungenderk1860deut/page/n245.

[10] Matteo Bottin, Giovanni Boschetti, and Giulio Rosati. "Optimizing Cycle Time of Industrial Robotic Tasks with Multiple Feasible Configurations at the Working Points". In: *Robotics* 11.1 (2022), p. 16. DOI: 10.3390/robotics11010016.

[11] Nicolaas Govert de Bruijn. "A combinatorial problem". In: *Proceedings of the Koninklijke Nederlandse Akademie van Wetenschappen* 49 (1946), pp. 758–764.

[12] Marcelo Campos et al. "An exponential improvement for diagonal Ramsey". In: *arXiv preprint* (2023). URL: https://arxiv.org/abs/2303.09521.

[13] Arthur Cayley. "On the theory of the analytical forms called trees". In: *Philosophical Magazine* 13 (1857), pp. 172–176. URL: https://rcin.org.pl/Content/173708/PDF/WA35_185808_12807-3_art-45.pdf.

[14] Gregory Chaitin et al. "Register allocation via coloring". In: *Computer Languages* 6 (1 1981), pp. 47–57. DOI: 10.1016/0096-0551(81)90048-5.

[15] Guantao Chen and Xingxing Yu. "A note on fragile graphs". In: *Discrete Mathematics* 249 (1 2002), pp. 41–43. DOI: 10.1016/S0012-365X(01)00226-6.

[16] Václav Chvátal. “Tough graphs and Hamiltonian circuits”. In: *Discrete Mathematics* 5 (1973), pp. 215–228. DOI: 10.1016/0012-365X(73)90138-6.

[17] Václav Chvátal and Pál Erdős. “A note on Hamiltonian circuits”. In: *Discrete Mathematics* 2 (2 1972), pp. 111–113. DOI: 10.1016/0012-365X(72)90079-9.

[18] Václav Chvátal and Peter Hammer. “Aggregation of inequalities in integer programming”. In: *Annals of Discrete Mathematics* 1 (1977), pp. 145–162.

[19] John H. Conway. *Five $1,000 Problems (Update 2017)*. 2017. URL: https://oeis.org/A248380/a248380.pdf (visited on 12/21/2025).

[20] K. Coolsaet, S. D’hont, and J. Goedgebeur. *House of Graphs 2.0: A database of interesting graphs and more*. 2023. DOI: 10.1016/j.dam.2022.10.013. URL: https://houseofgraphs.org.

[21] Daniel Cranston and Landon Rabern. “Brooks’ Theorem and Beyond”. In: *Journal of Graph Theory* 80 (3 2015). DOI: 10.1002/jgt.21847.

[22] Jochen W. Deuerlein. “Decomposition Model of a General Water Supply Network Graph”. In: *Journal of Hydraulic Engineering* 134 (6 2008). DOI: 10.1061/(ASCE)0733-9429(2008)134:6(822).

[23] Gabriel Dirac. “Short proof of Menger’s graph theorem”. In: *Mathematika* 13 (1 1966), pp. 42–44. DOI: 10.1112/S0025579300004162.

[24] Gabriel Dirac. “Some theorems on abstract graphs”. In: *Proceedings of the London Mathematical Society* 3.1 (1952), pp. 69–81. DOI: 10.1112/plms/s3-2.1.69.

[25] Jack Edmonds and Richard M. Karp. “Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems”. In: *Journal of the ACM* 19 (2 1972), pp. 248–264. DOI: 10.1145/321694.321699.

[26] Andrzej Ehrenfeucht, Vance Faber, and Henry Kierstead. “A new method of proving theorems on chromatic index”. In: *Discrete Mathematics* 52 (2–3 1984). DOI: 10.1016/0012-365X(84)90078-5.

[27] David Eppstein. *Twenty-one Proofs of Euler’s Formula: $V - E + F = 2$*. URL: https://ics.uci.edu/~eppstein/junkyard/euler/ (visited on 12/21/2025).

[28] Pál Erdős. “On an extremal problem in graph theory”. In: *Colloquium Mathematicum* 13 (1964), pp. 251–254. DOI: 10.4064/cm-13-2-251-254.

[29] Pál Erdős. “Some remarks on the theory of graphs”. In: *Bulletin of the American Mathematical Society* 53.4 (1947), pp. 292–294. DOI: 10.1090/S0002-9904-1947-08785-1.

[30] Pál Erdős and Tibor Gallai. “Gráfok előírt fokszámú pontokkal”. In: *Matematikai Lapok* 11 (1960), pp. 264–274. URL: https://www.renyi.hu/~p_erdos/1961-05.pdf.

[31] Abdol Esfahanian and S. L. Hakimi. “On computing the connectivities of graphs and digraphs”. In: *Networks* 14 (2 1984), pp. 355–366. DOI: 10.1002/net.3230140211.

[32] Leonhard Euler. “Elementa doctrinae solidorum”. In: *Novi Commentarii academiae scientiarum Petropolitanae* 4 (1758), pp. 109–140. URL: https://scholarlycommons.pacific.edu/euler-works/230/.

[33] P.J. Federico. *Descartes on Polyhedra: A Study of the “De Solidorum Elementis”*. Sources in the History of Mathematics and Physical Sciences. Springer, 1982. ISBN: 978-0-387-90760-4.

[34] L. R. Ford Jr. and D. R. Fulkerson. “Maximal Flow Through a Network”. In: *Canadian Journal of Mathematics* 8 (1956), pp. 399–404. DOI: 10.4153/CJM-1956-045-5.

[35] Robert Frucht. “Graphs of Degree Three with a Given Abstract Group”. In: *Canadian Journal of Mathematics* 1 (4 1949), pp. 365–378. DOI: 10.4153/CJM-1949-033-6.

[36] Martin Gardner. “M-Pire Maps”. In: New York, NY: Springer New York, 1997, pp. 85–100. ISBN: 978-0-387-30389-5.

[37] Martin Gardner. “The Five Platonic Solids”. In: New York, NY: Cambridge University Press, 2008, pp. 1–10. ISBN: 978-0-521-75610-5.

[38] Georges Gonthier. “Formal Proof—The Four-Color Theorem”. In: *Notices of the American Mathematical Society* 55 (11 2008), pp. 1382–1393.

[39] Ronald Graham and Daniel Kleitman. “Increasing paths in edge ordered graphs”. In: *Period. Math. Hungar.* 3 (1973). Collection of articles dedicated to the memory of Alfréd Rényi, II, pp. 141–148. ISSN: 0031-5303. DOI: 10.1007/BF02018469.

[40] Robert Greenwood and Andrew Gleason. “Combinatorial relations and chromatic graphs”. In: *Canadian Journal of Mathematics* 7 (1955), pp. 1–7. DOI: 10.4153/CJM-1955-001-4.

[41] Aubrey de Grey. “The chromatic number of the plane is at least 5”. In: *arXiv preprint* (2018). URL: https://arxiv.org/abs/1804.02385.

[42] Herbert Grötzsch. “Zur Theorie der diskreten Gebilde, VII: ein Dreifarbenansatz für dreikreisfrei Netze auf der Kugel”. In: *Wissenschaftliche Zeitschrift der Martin-Luther-Universität Halle-Wittenberg: Mathematisch-naturwissenschaftliche Reihe* 8 (1959), pp. 109–120.

[43] Parth Gupta et al. “Optimizing the CGMS upper bound on Ramsey numbers”. In: *arXiv preprint* (2024). URL: https://arxiv.org/abs/2407.19026.

[44] Richard Guy. “The strong law of small numbers”. In: *The American Mathematical Monthly* 95 (8 1988), pp. 697–712. DOI: 10.1080/00029890.1988.11972074.

[45] András Gyárfás, Jeno X. Lehel, and Zsolt Czap Tuza. “Clumsy packing of dominoes”. In: *Discrete Mathematics* 71.1 (1988), pp. 33–46. DOI: 10.1016/0012-365X(88)90028-3.

[46] S.L. Hakimi. “On Realizability of a Set of Integers as Degrees of the Vertices of a Linear Graph. I”. In: *Journal of the Society for Industrial and Applied Mathematics* 10.3 (1962), pp. 496–506. DOI: 10.1137/0110037.

[47] Philip Hall. “On Representatives of Subsets”. In: *Journal of the London Mathematical Society* 10.1 (1931), pp. 26–30. DOI: 10.1112/jlms/s1-10.37.26.

[48] Richard Hammack. *Book of Proof*. 2018. ISBN: 978-0-9894721-2-8. URL: https://richardhammack.github.io/BookOfProof/.

[49] Frank Harary. “The maximum connectivity of a graph”. In: *Proceedings of the National Academy of Sciences of the United States of America* 48 (1962), pp. 1142–1146. DOI: 10.1073/pnas.48.7.1142.

[50] Frank Harary and Robert Norman. “Some properties of line digraphs”. In: *Rendiconti del Circolo Matematico di Palermo* 9 (1960), pp. 161–168. DOI: 10.1007/BF02854581.

[51] Mor Harchol-Balter. *Performance Modeling and Design of Computer Systems*. Cambridge University Press, 2013. ISBN: 978-1-107-02750-3. URL: https://www.cs.cmu.edu/~harchol/PerformanceModeling/book.html.

[52] Václav Havel. “Poznámka o existenci konečných grafů”. In: *Časopis pro pěstování matematiky* 80.4 (1955), pp. 477–480. DOI: 10.21136/CPM.1955.108220.

[53] Yuan He et al. “On the reliability of large-scale distributed systems — A topological view”. In: *Computer Networks* 53 (12 2009), pp. 2140–2152. DOI: 10.1016/j.comnet.2009.03.012.

[54] Percy J. Heawood. “Map colour problem”. In: *Quarterly Journal of Pure and Applied Mathematics* 24 (1890), pp. 332–338.

[55] Samad Hedayat and Walter T. Federer. “On the nonexistence of Knut Vik designs for all even orders”. In: *The Annals of Statistics* 3.2 (1975), pp. 445–447. DOI: 10.1214/aos/1176343068.

[56] Alan Hoffman and Robert Singleton. “On Moore graphs with diameters 2 and 3”. In: *IBM Journal of Research and Development* 4 (5 1960), pp. 497–504. DOI: 10.1147/rd.45.0497.

[57] Derek Holton and John Sheehan. *The Petersen Graph*. Australian Mathematical Society Lecture Series. Cambridge University Press, 1993. ISBN: 978-0-521-43594-9.

[58] Hud Hudson. “Four Colors Do Not Suffice”. In: *The American Mathematical Monthly* 110 (5 2003), pp. 417–423. DOI: 10.1080/00029890.2003.11919979.

[59] Brad Jackson and Gerhard Ringel. “Solution of Heawood’s empire problem in the plane”. In: *Journal für die Reine und Angewandte Mathematik* 347 (1984), pp. 146–153. DOI: 10.1515/crll.1984.347.146.

[60] Paul C. Kainen. “A generalization of the 5-color theorem”. In: *Proceedings of the American Mathematical Society* 45 (3 1974), pp. 450–453. DOI: 10.2307/2039977.

[61] Michael Kleber and Ravi Vakil. “The best card trick”. In: *The Mathematical Intelligencer* 24 (2002), pp. 9–11. DOI: 10.1007/BF03025305.

[62] Dénes Kőnig. “Gráfok és mátrixok”. In: *Matematikai és Fizikai Lapok* 38 (1931), pp. 116–119.

[63] Dénes Kőnig. *Theorie der endlichen und unendlichen Graphen*. Vol. 72. American Mathematical Society, 2001.

[64] Dénes Kőnig. “Über Graphen und ihre Anwendung auf Determinantentheorie und Mengenlehre”. In: *Mathematische Annalen* 77 (1916), pp. 453–465. DOI: 10.1007/BF01456961.

[65] Joseph B. Kruskal. “On the shortest spanning subtree of a graph and the traveling salesman problem”. In: *Proceedings of the American Mathematical Society* 7.1 (1956), pp. 48–50. DOI: 10.2307/2033241.

[66] David Kullman. “The Utilities Problem”. In: *Mathematics Magazine* 52 (5 1979), pp. 299–302. DOI: 10.1080/0025570X.1979.11976807.

[67] Casimir Kuratowski. “Sur le problème des courbes gauches en topologie”. In: *Fundamenta Mathematicae* 15 (1930), pp. 271–283. DOI: 10.4064/fm-15-1-271-283.

[68] William Wallace Lee. *Math miracles*. Durham, N.C.: Seeman Printery, 1950.

[69] Ben P.C. Li and Michel Toulouse. “Maximum leaf spanning tree problem for grid graphs”. In: *Journal of Combinatorial Mathematics and Combinatorial Computing* 73 (2010), p. 181. URL: https://combinatorialpress.com/jcmcc-articles/volume-073/maximum-leaf-spanning-tree-problem-for-grid-graphs/.

[70] László Lovász and Michael D. Plummer. *Matching Theory*. North Holland, 1986. ISBN: 978-0-8218-4759-6.

[71] David C. Lovelace. *RPS-7*. 2003. URL: https://umop.com/rps7.htm (visited on 12/21/2025).

[72] Karl Menger. "Zur allgemeinen Kurventheorie". In: *Fundamenta Mathematicae* 10 (1927), pp. 96–115. DOI: 10.4064/fm-10-1-96-115.

[73] Mirka Miller and Jozef Širáň. "Moore graphs and beyond: a survey of the degree/diameter problem". In: *Electronic Journal of Combinatorics* (DS14 2013). DOI: 10.37236/35.

[74] Edward F. Moore. "The shortest path through a maze". In: *Proc. Internat. Sympos. Switching Theory 1957, Parts I,II*. The Annals of the Computation Laboratory of Harvard University, vols. XXIX, XXX. Harvard Univ. Press, Cambridge, MA, 1959, pp. 285–292.

[75] Leo Moser and William Moser. "Problems for Solution (P10)". In: *Canadian Mathematical Bulletin* 4 (2 1961), pp. 187–189. DOI: 10.1017/S0008439500025753.

[76] Jan Mycielski. "Sur le coloriage des graphs". In: *Colloquium Mathematicum* 3 (2 1955), pp. 161–162. DOI: 10.4064/cm-3-2-161-162.

[77] Clive Newstead. *An Infinite Descent into Pure Mathematics*. 2024. ISBN: 978-1-950215-00-3. URL: https://infinitedescent.xyz/.

[78] OEIS Foundation Inc. *Entry A000055 in the On-Line Encyclopedia of Integer Sequences*. 2026. URL: https://oeis.org/A000055.

[79] OEIS Foundation Inc. *Entry A000109 in the On-Line Encyclopedia of Integer Sequences*. 2026. URL: https://oeis.org/A000109.

[80] OEIS Foundation Inc. *Entry A002851 in the On-Line Encyclopedia of Integer Sequences*. 2026. URL: https://oeis.org/A002851.

[81] OEIS Foundation Inc. *Entry A380996 in the On-Line Encyclopedia of Integer Sequences*. 2026. URL: https://oeis.org/A380996.

[82] George Pólya. *How to Solve It*. Princeton University Press, 1945. ISBN: 978-0-691-16407-6.

[83] George Pólya. "Kombinatorische Anzalbestimmungen für Gruppen, Graphen, und chemische Verbindungen". In: *Acta Mathematica* 68.1 (1937), pp. 145–254. DOI: 10.1007/BF02546665.

[84] George Pólya. "Uber die "doppelt-periodischen" Lösungen des n-Damen-Problems". In: *W. Ahrens, Mathematische Unterhaltungen und Spiele* 1 (1921), pp. 364–374.

[85] Rob Pratt. *16 queens puzzle*. 2023. URL: https://puzzling.stackexchange.com/a/122418/47819 (visited on 12/21/2025).

[86] Princeton University Math Club. *2019 Princeton University Mathematics Competition*. 2019. URL: https://pumac.princeton.edu/archives/#2019 (visited on 12/21/2025).

[87] Heinz Prüfer. "Neuer Beweis eines Satzes über Permutationen". In: *Arch. Math. Phys* 27 (1918), pp. 742–744.

[88] Stanisław Radziszowski. "Small Ramsey numbers". In: *Electronic Journal of Combinatorics* (DS11 2024). DOI: 10.37236/21.

[89] Frank Ramsey. “On a problem of formal logic”. In: *Proceedings of the London Mathematical Society* 2 (1 1930), pp. 264–286. DOI: 10.1112/plms/s2-30.1.264.

[90] Neil Robertson et al. “The Four-Colour Theorem”. In: *Journal of Combinatorial Theory, Series B* 70 (1 1997), pp. 2–44. DOI: 10.1006/jctb.1997.1750.

[91] Claude Shannon. “A theorem on coloring the lines of a network”. In: *Journal of Mathematics and Physics* 28 (1–4 1949), pp. 148–152. DOI: 10.1002/sapm1949281148.

[92] SimpleMaps.com. *Free GIS Maps of Switzerland*. Licenced under CC BY 4.0. 2010–2025. URL: https://simplemaps.com/gis/country/ch (visited on 12/21/2025).

[93] Emanuel Sperner. “Ein Satz über Untermengen einer endlichen Menge”. In: *Mathematische Zeitschrift* 27.1 (1928), pp. 544–548. DOI: 10.1007/BF01171114.

[94] Ernst Steinitz. *Über die Konstruction der Configurationen n_3*. Druck v. Dr. R. Galle, 1894.

[95] David Sumner. “Graphs with 1-Factors”. In: *Proceedings of the American Mathematical Society* 42 (1 1974), pp. 8–12. DOI: 10.2307/2039666.

[96] Dan Thomasson. *Knight’s Tour Golden Star*. 2002. URL: http://danthomasson.com/kt-golden-star.html (visited on 12/21/2025).

[97] Richard J. Trudeau. *Introduction to Graph Theory*. Dover Publications, Inc., 1993. ISBN: 978-0-486-67870-2.

[98] William Thomas Tutte. “Matroids and Graphs”. In: *Transactions of the American Mathematical Society* 90 (3 1959), pp. 527–552. DOI: 10.2307/1993185.

[99] William Thomas Tutte. “On Hamiltonian circuits”. In: *Journal of the London Mathematical Society* 21 (2 1946), pp. 98–101. DOI: 10.1112/jlms/s1-21.2.98.

[100] William Thomas Tutte. “The Factorization of Linear Graphs”. In: *Journal of the London Mathematical Society* 1.2 (1947), pp. 107–111. DOI: 10.1112/jlms/s1-22.2.107.

[101] Oswald Veblen. “An Application of Modular Equations in Analysis Situs”. In: *Annals of Mathematics, Second Series* 14.1 (1912), pp. 86–94. DOI: 10.2307/1967604.

[102] Vadim Vizing. “On an estimate of the chromatic class of a p-graph”. In: *Diskretnyi Analiz* 3 (1964), pp. 25–30.

[103] Klaus Wagner. “Über eine Eigenschaft der ebenen Komplexe”. In: *Mathematische Annalen* 114 (1937), pp. 570–590. DOI: 10.1007/BF01594196.

[104] Douglas B. West. *Introduction to Graph Theory*. Prentice Hall, 1996. ISBN: 978-0-13-227828-7.

[105] Hassler Whitney. “Congruent Graphs and the Connectivity of Graphs”. In: *American Journal of Mathematics* 54 (1 1932), pp. 150–168. DOI: 10.2307/2371086.

[106] Hassler Whitney. “Non-separable and planar graphs”. In: *Transactions of the American Mathematical Society* 34 (1932), pp. 339–362. DOI: 10.1090/S0002-9947-1932-1501641-2.

[107] Robin J. Wilson. *Four colors suffice: how the map problem was solved*. Princeton, NJ: Princeton University Press, 2002. ISBN: 978-0-691-11533-7.

[108] Peter Winkler. “Puzzled: Solutions and sources”. In: *Communications of the ACM* 51.9 (2008), pp. 103–103. DOI: 10.1145/1378727.1389570.

E Index

•, *see* contraction
⊕, *see* symmetric difference
$\partial_G(S)$, *see* edge boundary

acyclic graph, *see* forest
adjacent, **18**, **103**
algorithm, 55–58, 91–94, 117–123, 132–135, 146–149, 154–162, 165–169, 180–185, 194, 197–205, 209, 254–258, 269–271, 294–295, 347–351, 393, 413–421, 430, 435–438
$\alpha(G)$, *see* independence number
α/β-path, *294*, 294–295
alphabet, 288–291
$\alpha'(G)$, *see* matching number
alternating cycle, *see* alternating path
alternating path, **196**, 194–197, 199–202, 224–226, 294–295
American Invitational Mathematics Examination, 176, 342
angle defect, *340*, 339–341
arc, **107**, 106–110, 121–123, 165–174, 283–291, 407–423
arc cut, *see* edge cut
Archimedean solid, *339*, 339–342
Asian Pacific Mathematics Olympiad, 150, 265
augmenting path, **196**, 194–205, 209, 224–226, *416*, 413–421, 423
automorphism, 32–39, 44, 45, 80–83, 238, 243, 379, 382–386, 390
average degree, 64, 68–71, 238, 347–349, 352–355

$B(k, n)$, *see* de Bruijn digraph
backward arc, *416*, 413–421
balanced vertex, *122*, 121–123, 288–291
barbell graph, 241
barycentric subdivision, *307*, 307–309, 453
base case, **444**
Bay Area Mathematical Olympiad, 149
$\beta(G)$, *see* vertex cover number
BFS, *see* breadth-first search
bijection, 32–36, 39–41, 53, 151–154
binary tree, 64
binomial coefficient, 18–20, 211, 216, 258–261, 385
bipartite graph, 26–27, 125, **180**, 180–186, 188–191, 193–194, 197–218, 220–222, 226–230, 241, 247, 269, 272, 291–294, 296, 315, 321, 327–330, 342, 351–352, 356, 364, 379, 393–395, 401, 424
bipartition, **180**, 180–185, 189, 191, 197–218, 221, 269, 296, 327–329, 379, 393–395, 424
blossom algorithm, 220
book graph, 241
bottleneck arc, *420*, 420–421
boundary walk, **303**, 302–305, 309–313, 321, 369–370
breadth-first search, 55–58, 111, 180–183, 197–199, 414, 419, 435
bridge, **130**, 130–137, 142–143, 241, 302–306, 309–313, 351, 360, 374, 376
bridges of Königsberg, 113–115, 194
British Mathematical Olympiad, 87, 88, 125, 192, 248, 297

C_n, *see* cycle graph
$\overrightarrow{C_n}$, *see* directed cycle graph
capacity, *409*, *412*, 407–423
card trick, 210–213, 218
Cayley, Arthur, 128, 159
centroid, 307
chess, 24–26, 29, 178–183, 191, 235–237, 249–251, 283–286
$\chi(G)$, *see* chromatic number
$\chi'(G)$, *see* edge chromatic number
chromatic index, *see* edge chromatic number
chromatic number, 15–18, **268**, 266–282, 291,

345–355, 361, 378, 386
chromatic number of the plane, 281
Chvátal, Václav, 242, 243, 403, 450
$\mathrm{Ci}_n(d_1, \ldots, d_k)$, *see* circulant graph
circulant graph, **32**, 30–36, 42–45, 58, 75–78, 80, 124, 261–263, 280, 300, 390, 428, 440
claw-free graph, 233, 296
clique, 23–24, 43, **251**, 249–264, 271–279, 282, 284, 286, 345–347, 362
clique number, **251**, 249–265, 269, 271–281, 292, 345–347, 362, 390, 435
closed walk, **54**, 53–55, 66–68, 110, 111, 113–121, 124, 125, 183–185, 286–291, 302–305, 362–363
co-bipartite graph, 282
color class, **268**, 266–269, 271–273, 282, 291–294
k-colorable graph, **268**, 266–273, 377
coloring, 15–18, 29, 146–149, **268**, 266–282, 328, 343–356, 361, 386, 435
comparable sets, 215–218, 457
complement, **19**, 18–20, 24, 78–81, 87, 226, 247, 249–254, 265, 296, 441, 456
complete k-partite graph, 247, 264
complete bipartite graph, **181**, 180–183, 232, 246, 285, 300, 302, 321–326, 373
complete graph, **43**
condensation, *172*, 171–175
congruent modulo m, *see* modular arithmetic
connected component, **50**
connected graph, **48**
k-connected graph, 352, *362*, 362–374, **377**, 376–386, 390, 402, 405, 406, 449
connectivity, 78, **377**, 376–382, 389–391, 402–406, 423, 435
  $s-t$ connectivity, **383**, 382–390, 392–402, 423
contraction, 313, **324**, 324–326, 350–351
contrapositive, **427**
convex polyhedron, *333*, 331–333, 339–342
Conway's 99-graph problem, 71, 265
copy of a graph, **43**, 42–45, 68, 79, 110, 150, 223, 232, 233, 241, 251, 255, 265, 271–273, 275–279, 293, 296, 320, 323, 350, 372, 379–382, 448, 452–456
cost, 25, **133**, 132–137, 149
covered by a matching, **186**, 185–186, 194–206, 210
cube graph, 53–58, **60**, 60–61, 63, 70, 127–129, 136, 137, 232, 247, 280, 331–338, 342, 365, 366, 370, 379–382, 436, 437
cut, *see* vertex cut
cut edge, *see* bridge
cut vertex, 360–369, 372–374, 376–379
cycle, **43**
  represented by a walk, **54**
cycle decomposition, *117*, 117–124, 449
cycle graph, **43**, 42–44, 53–58, 78–81, 104, 117, 131, 205, 241–243, 247, 264, 279, 281, 284, 312, 330, 430, 440, 447–448, 456

de Bruijn digraph, *289*, 288–291, 453
de Bruijn sequence, *289*, 288–291, 296
decomposition, 117–119, 226–230
$\deg(x)$, *see* degree
$\deg^-(x)$, *see* indegree
$\deg^+(x)$, *see* outdegree
degree, **61**, **105**
degree (geometry), 331, 340
degree sequence, 41, **73**, 73–75, 88–100, 104–105, 111, 136, 149, 150, 243–246, 248, 424
degree sum formula, *see* handshake lemma
degree/diameter problem, 83–86
deletion sequence, *155*, 154–162
$\Delta(G)$, *see* maximum degree
$\delta(G)$, *see* minimum degree
Descartes, René, 340
diameter, 65, *83*, 83–87, 98, 136, 435, 455–456
digraph, *see* directed graph
Dirac's fan lemma, *402*, 402–406
Dirac, Gabriel, 243, 393, 402
direct proof, **427**
directed
  acyclic graph, **166**, 165–176
  cycle graph, *110*, 122, 169
  edge, *see* arc
  graph, **107**, 106–112, 121–123, 165–175, 283–291, 401–402, 407–409, 411, 414, 421–424
  multigraph, 107
  path graph, *110*, 169

discharging method, 340–341
disjoint union, *see* union
distance, **56**, 55–59, 61, 65, 83–86, 183–185, 224, 356, 374, 406, 418–421, 433–435, 455–456
dodecahedron, 70, 238–240, 331–338, 341
double factorial, 154
dual graph, 313, **336**, 336–338, 341, 342, 344, 351–352

ear, **365**, 364–373
ear decomposition, 117, **366**, 364–374, 377, 379, 390, 449
edge, **17**
edge boundary, **378**, 376–382, 411, 422
edge chromatic number, 291–296, 378
edge coloring, 291–295, 297, 435
edge connectivity, **378**, 376–382, 390, 391, 423
  $s-t$ edge connectivity, **383**, 382–389, 399–401, 423
edge cut, **378**, 376–386, 391
  $s-t$ arc cut, 411, 421–423
  $s-t$ edge cut, **383**, 382–389, 392, 399–401
edge swap, *95*, 94–99
k-edge-connected graph, 374, **378**, 376–379
edge-disjoint, **384**, 382–389, 392, 399–401, 421–423
eight queens puzzle, 249–251, 265, 283–286
empty graph, 78–81, 92, 251
encoding scheme, *152*, 151–160
endpoint, **18**
equivalence class, *see* equivalence relation
equivalence relation, *51*, 51–53, 59, 151–154, 162–163, 431, 441
Erdős, Pál, 68, 100, 263, 403
Euler tour, 48, **115**, 113–125, 230, 286–291, 296
Euler walk, **115**, 113–117, 121–123
Euler's formula, 305–314, 316–321, 333–335, 339–341, 449
Euler's number, 163
Euler, Leonhard, 115, 194, 305
Eulerian graph, **115**, 113–117, 119–124, 230, 287, 440
even vertex, *115*, 115–123, 125, 146–149
existential quantifier, **428**
extremal principle, **438**

face, **302**, 302–313, 316–321, 330, 333–338, 344, 369–370
face length formula, **304**, 302–305, 307–309, 313, 314, 316–318, 321, 333–339
1-factor, *see* perfect matching
2-factor, *237*, 424
factorial, 179, 212, 218, 307–309
1-factorization, 117, **227**, 226–232, 291–294
family of graphs, 30–32, 42–44, 60–61, 75–78, 81–86, 110, 138, 145, 209, 235, 279, 281, 289, 450
$s-T$ fan, *402*, 402–405
feasible flow, *409*, 409–421, 423, 424
Fibonacci sequence, 456
five rooms puzzle, 113–115
flow, *409*, 409–423
flower garden problem, 138–140, 143–145, 149
Ford–Fulkerson algorithm, 413–424
forest, **143**, 142–143, 166, 242, 297, 312, 321
forward arc, *416*, 413–421
four color theorem, **346**, 345–347, 351–352
fragile graph, 390
fragment, *327*, 327–329
Frucht graph, 80

$G(n,m)$, *see* grid graph
girth, *321*, 321, 329
graph, **17**
graph invariant, **39**, 39–44, 65–66, 193–194, 199, 249–254, 266–269, 283–286, 301–302, 364, 376–379
graph of adjacencies, 268, 313, 343–356
graph property, *see* graph invariant
graphic sequence, **74**, 73–75, 88–100, 105, 111
Gray code, 247
greatest common divisor, 59
greedy algorithm, 132–135, 187, 188, 199, 202, 254–258, 264, 269–271, 274–275, 280, 281, 347–349, 354, 364–370, 413–418, 424
grid, 23–24, 28, 29, 138–140, 178–180, 191, 219, 264, 266–269, 358–360
grid graph, **138**, 138–140, 143–145, 178–183, 187, 196, 325, 330, 359

Grötzsch graph, 279

$H_{n,r}$, *see* Harary graph
Hakimi, S. L., 94, 390
Hall's condition, *see* Hall's theorem
Hall's theorem, 62, **207**, 206–218, 220–222, 232, 406, 423
Hamilton cycle, 55, **236**, 235–247, 286–291, 293, 300, 314, 327–329, 346, 351–352, 355, 363, 364, 368, 403–405
Hamilton path, 24–26, **236**, 235–237, 244–245, 355
Hamiltonian graph, **236**, 235–248, 363, 364, 403–406, 440
handshake lemma, **62**, 61–65, 68–70, 75, 89, **108**, 104–109, 122, 141, 143–145, 148, 191, 333–341, 347, 354, 438, 449, 454
Harary graph, *78*, 75–78, 86, 389
Harary, Frank, 78, 284
Havel–Hakimi theorem, **94**, 94–99
Hoffman–Singleton graph, 86, 87
House of Graphs, 319
hypercube graph, 55, **60**, 60–65, 71, 76, 124, 136, 183–185, 215–218, 247, 382–386, 389, 452–456

icosahedron, 70, 331–338, 341, 349, 355, 406
Icosian game, 238–240
icosidodecahedron, 342
identity automorphism, *36*, 80
improper coloring, *268*, 266–269, 277–278
improper ear decomposition, 374
incident, **18**, **103**
increasing walk, 48, *230*, 230–232
indegree, **107**, 106–109, 121–123, 167, 175, 291, 409, 422
independence number, **250**, 249–265, 269, 271–273, 280, 285, 373, 378, 403–406, 435
independent set, 23–24, 29, 178–180, **250**, 249–264, 267, 271–274, 279, 283–286, 291, 294, 296, 390, 403–405
induced subgraph, **42**, 42–45, 51–53, 71, 169, 216, 233, 251, 259, 296, 297, 361, 397–399, 441
induced subgrpaph, 50
induction hypothesis, **444**
induction step, **444**
inductive definition, 369–371, **450**
integer flow, *418*, 418–424
internal vertex, **365**, 364–369, 371–374, 382–386, 416
internally disjoint, 373, **384**, 382–390, 392–405
International Mathematical Olympiad, 59, 71, 195
intersection of graphs, 19
interval graph, **252**, 252–254, 274–275
invariant
  of a graph, *see* graph invariant
  of an algorithm, 199–202, 294–295, **437**, 436–438
inverse function, 32–36, 39–41, 382–386
isolated vertex, **62**, 68, 78, 118–123, 140, 144, 147, 206
isomorphism, **33**
isomorphism game, 36–41
isomorphism invariant, *see* graph invariant

Jordan curve theorem, 310

$K(n,k)$, *see* Kneser graph
K_n, *see* complete graph
$K_{m,n}$, *see* complete bipartite graph
$\kappa(G)$, $\kappa_G(s,t)$, *see* connectivity
$\kappa'(G)$, $\kappa'_G(s,t)$, *see* edge connectivity
Kneser graph, *82*, 81–83, 87
knight graph, *235*, 235–237, 241, 246
knight's tour, 24–26, 29, 235–237, 265
Knut Vik design, 251, 264
Kőnig's theorem, 190, **194**, 193–194, 197–202, 205–209, 364, 386–389, 393–395, 399, 406, 422
Kőnig, Dénes, 194, 229, 292
Kőnig-type graph, *393*, 393–396, 399, 401
Kruskal's algorithm, 133
Kuratowski's theorem, **322**, 322–324, 326, 344, 364

$L(G)$, *see* line graph
labeled graph, 151–154
Latin square, 219
leaf, **62**, 64, 138–140, 143–149, 154–162, 164, 167, 207, 241, 314, 360, 367, 449
$\text{len}(F)$, *see* length of a face
length

of a cycle, **55**, 58, 66–68, 81–84, 87, 183–185, 191, 316–318, 321, 328, 373, 403–405, 429
of a face, **304**, 302–305, 307–309, 313, 314, 316–321, 330, 333–338
of a path, **55**, 110, 165–169, 194–199, 373, 418–421
of a walk, 20–22, **55**, 55–59, 83–86, **104**, 110, 111, 183–185, 230–231, 304, 455–456
line digraph, **285**, 283–291
line graph, **284**, 283–288, 291, 296, 374, 378, 391, 399–401, 453
logic grid puzzle, 26–27
lollipop graph, 241
loop, **103**, 102–106, 111, 118, 129, 131, 209, 316–317, 324, 336, 401

M_k, *see* Mycielski graph
M-pire, *353*, 352–355
Möbius ladder graph, 45
map coloring, 15–18, 28
matching, 178–180, **185**, 185–191, 193–204, 206–208, 213–218, 220–222, 224–230, 283–286, 291–295, 364, 386–389, 393–395, 416, 435, 456
matching number, **193**, 193–194, 202, 205, 232, 252–254, 265, 285, 364, 378, 393–395
Mathematica, 23
maximal planar graph, *319*, 319–320, 352
maximum degree, **62**, 61–65, 70, 83–86, 164, 193, 209, 254–256, 269–271, 280, 286, 291–296, 347–349, 433–435, 456
maximum flow, 407–424
maximum/maximal, *187*, 187–188, 191, 254–256
MCST, *see* minimum-cost spanning tree
Menger's theorem, 389, **393**, **399**, 392–402, 406, 421–424
metric space, 59
middle layer, 215–218
minimum degree, **62**, 61–70, 117–119, 150, 191, 193, 205, 218, 243–246, 256, 296, 347–355, 367, 379–382, 386, 433, 447–448, 456
minimum-cost spanning tree, *133*, 132–137
minimum/minimal, *187*, 187–188, 378
minor, *325*, 324–326, 329
Möbius strip, 45, 355
modular arithmetic, **31**, 30–36, 75–78, 226–230, 249–251, 262, 264
modulo m, *see* modular arithmetic
monovariant, 437
Moore bound, 83–87
Moore graph, 87
Moore, Edward, 58, 86
Moser spindle, 281
multigraph, 43, **103**, 102–106, 111, 113–117, 119, 129, 131, 208–210, 293, 296, 302, 306, 314, 316–317, 324, 336–338, 342, 390, 401
multiset, 74
Mycielski graph, *279*, 275–279, 453
Mycielskian, *275*, 275–279, 453

$N_G(S)$, *see* neighborhood
necessary and sufficient, 73–75, 115–117, 119–123, 241–246, 316–318, **427**
necessary condition, *see* necessary and sufficient
neighbor, **18**
neighborhood, **207**, 206–210, 213–215, 247, 254, 457
net degree, 108
network, *409*, 409–424
network cut, *412*, 411–423
NP-complete, 235

octahedron, 70, 331–338, 342
odd component, *221*, 220–222, 224–227, 232
odd cycle, *183*, 183–185, 272
odd vertex, 64, *115*, 115–117, 121–124, 146–149
OEIS, 80, 163, 307–309, 319
$\omega(G)$, *see* clique number
order of a graph, 40
orientation, 107
outdegree, **107**, 106–110, 121–123, 167, 175, 291, 409, 422
outer face, **302**, 302–305, 307–309, 314, 320, 330, 341, 369
overlap graph, 300, *328*, 327–330

P_n, *see* path graph
$\overrightarrow{P_n}$, *see* directed path graph
pairing strategy, 213–215, 218

Paley graph, 262
pancyclic graph, 341
parallel edge, **103**, 102–106, 118, 129, 131, 209, 316–317, 324, 336, 401
partition, 51, 52, 170, 266–269, 271–273
Pascal's identity, 260–261
path, **43**
  represented by a walk, **48**
path graph, **43**, 42–44, 46–48, 55–58, 69, 87, 131, 143–145, 162–164, 284, 441, 447, 456
perfect matching, 26–27, 125, **186**, 185–190, 204–213, 218, 220–232, 237, 247
Petersen graph, 81–86, 238–243, 291–294, 329, 390, 405
planar graph, 41, **301**, 301–307, 309–314, 316–338, 343–356, 361, 364, 369–370, 440
plane embedding, **301**, 299–338, 341, 344, 351–352, 361, 364, 369–370, 449
Platonic solid, 70, *332*, 331–338, 342
Pólya, George, 163, 251, 426
polygon, *309*, 309–313, 331–333
polygonal curve, *309*, 309–313
polyhedron, 305, **331**, 331–333
Prim's algorithm, 133
Princeton University Mathematics Competition, 436
prism, 341
product principle, 151–154, 179, 358–360
proof by contradiction, **427**
proof by contrapositive, **427**
proper coloring, *see* coloring, 251
proper edge coloring, *see* edge coloring
Prüfer code, **158**, 157–164
pure loop, 28

Q_n, *see* hypercube graph
quadratic residue, 262
queen graph, *250*, 249–252

$R(k, l)$, *see* Ramsey number
Ramsey graph, 261–264
Ramsey number, *258*, 258–264
random graph, 98, 261–263, 271–273, 438
ray casting algorithm, 310
reachability, 51–53, 171–173, 431–433
recursive construction, **452**
reflexive relation, *51*, 51–53
register allocation, 274–275
regular graph, **75**, 75–86, 111, 125, 136, 137, 151–154, 205, 208–213, 226–230, 232, 265, 292, 296, 297, 329, 333–335, 342, 352, 372, 374, 389
residual capacity, *416*, 413–418
residual network, *416*, 413–421, 423, 436
reverse-delete algorithm, *133*, 132–135
rook graph, *283*, 283–286
rook polynomial, 191
round-robin tournament, 226–230

S_n, *see* star graph
saturated graph, *223*, 223–226
shortest path, 20–22, 55–58, 111
side, *see* bipartition
similar vertices, 45
simple graph, **103**, 111, 118, 129, 131, 209, 316–317
simplification, **103**, 210, 302
sink, *109*, 106–111, 165–169, *409*, 409–423
size of a graph, 40
skeleton graph, *332*, 331–333, 337, 340, 341, 406
snub cube, 342
source, *109*, 106–111, 165–168, *409*, 409–423
spanning cycle, *see* Hamilton cycle
spanning subgraph, **42**, 42–45, 125, 127–129, 138–140, 143–149, 185, 211, 226, 235–237, 287, 368, 456
spanning tree, 127–129, 132–140, 143–149, 164, 362, 372
spherical embedding, 333, 341
star battle, 28
star graph, **145**, 143–145, 162–164, 181, 233, 254–256, 296, 347
state graph, 20–22, 46–51
Stirling's formula, 163
strong duality, *387*, 386–389, 392–393
strong induction, **446**
strongly connected component, *169*, 169–174
strongly connected digraph, *169*, 169–171, 390
subdivision, **322**, 322–324, 326, 329, 344, 364, 373
subgraph, **42**, 42–55, 68–70, 119–121, 127–129, 133, 169–171, 180, 194–197, 244,

296, 322, 324–326, 360–362, 364–369, 373, 383, 416, 422
substring, *288*, 288–291, 296
Sudoku, 266–269
sufficient condition, *see* necessary and sufficient
symmetric difference, 148, **194**, **195**, 194–197, 224–226
symmetric relation, 47, *51*, 51–53, 103, 431–432
symmetry, *see* automorphism

tetrahedron, 70, 331–341
three utilities problem, 299–300
threshold graph, *450*, 450–452, 456, 457
tic-tac-toe, 213–215
tiling, 23–24, 29, 178–183, 187–189, 191, 204
topological order, **168**, 168–169, 174, 175
total degree, 108
tough graph, *241*, 241–243, 247, 363, 364
Towers of Hanoi, 20–22, 28, 46, 442–446, 456
traceable graph, 236, 247
transitive relation, *51*, 51–53
tree, **128**, **141**, 127–150, 154–168, 175, 183–185, 269–271, 280, 281, 314, 341, 347, 351–352, 356, 360, 438, 449
triangle-free graph, 277
triangulation, **319**, 319–320, 351–352
trivial automorphism, *see* identity automorphism
truncated icosidodecahedron, 342
truncated tetrahedron, 339–341
Tutte graph, *352*, 351–352, 355
Tutte set, *222*, 220–226, 231, 232
Tutte's theorem, **222**, 220–226, 232, 233
Tutte, William, 222, 327, 352
Tutte–Berge formula, 220, 232

underlying graph, **107**, 109, 112, 123, 169, 175, 390
union, **19**, 18–20, 48–53, 66–68, 78–81, 150, 169–171, 173, 215–218, 261, 294, 356, 364–369, 457
unique encoding scheme, *152*, 151–160
unit distance graph, 281
United States of America Mathematical Olympiad, 232
universal quantifier, **429**
unlabeled graph, 79, 151–154
up to isomorphism, **78**, 78–81, 88–90, 128, 145, 149, 151–154, 162–163, 289, 319, 341, 342

vacuously true, **429**
valid encoding, *152*, 151–160
value of a flow, *410*, 409–423
vertex, **17**
vertex connectivity, *see* connectivity
vertex cover, 188–191, 193–194, 197–204, 206–209, 253, 282, 364, 386–389, 393–395, 435
vertex cover number, **193**, 193–194, 202, 205, 206, 252–254, 393–395
vertex cut, **376**, 376–386, 391
  $s-t$ cut, **383**, 382–389, 392–402
Vizing's theorem, **294**, 291–296, 349
volcano graph, *202*, 202–204, 313

Wagner's theorem, **326**, 324–326
walk, **48**, **104**, **110**
weak duality, *387*, 386–389, 392–393
weakly connected, *123*, 121–123, 169, 288–291
weighted graph, 24–26, **133**, 132–136, 149, 230–231, 407–409
wheel graph, 281
Whitney, Hassler, 363, 380
William Lowell Putnam Mathematical Competition, 59, 219, 330
word ladder, 28