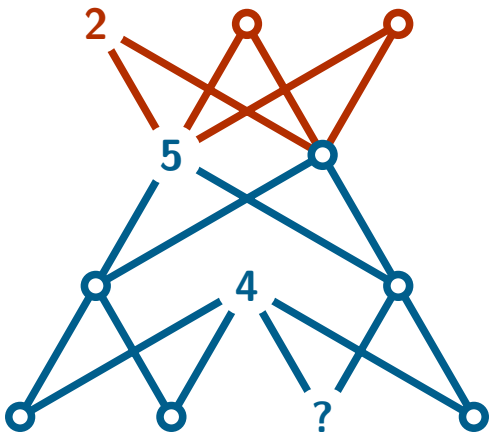
Mikhail Lavrov

Start Doing Graph Theory

Part I: The Beginning



available online at https://vertex.degree/

Contents

About this document **3**
- Where it's from . . . 3
- What's inside . . . 3
- The cover . . . 3
- The license . . . 3

1 Modeling problems with graphs **4**
- 1.1 Coloring Switzerland . . . 4
- 1.2 The seven dwarfs . . . 7
- 1.3 The Towers of Hanoi . . . 9
- 1.4 A tiling puzzle . . . 12
- 1.5 Taking photos efficiently . . . 13
- 1.6 Match the cars and drivers . . . 15
- 1.7 Practice problems . . . 16

2 Isomorphisms and subgraphs **19**
- 2.1 Circulant graphs . . . 19
- 2.2 Are these the same? . . . 21
- 2.3 The isomorphism game . . . 25
- 2.4 Graph invariants . . . 28
- 2.5 Subgraphs and some common small graphs . . . 31
- 2.6 Practice problems . . . 33

3 Walks, paths, and cycles **35**
- 3.1 Walks and paths . . . 35
- 3.2 Connected graphs . . . 37
- 3.3 Equivalence relations . . . 40
- 3.4 Closed walks and cycles . . . 42
- 3.5 Lengths of walks . . . 44
- 3.6 Practice problems . . . 47

4 The degree of a vertex **49**
- 4.1 The hypercube graphs . . . 49
- 4.2 The handshake lemma . . . 50
- 4.3 Degrees and connectedness . . . 54
- 4.4 Degrees and cycles . . . 55
- 4.5 Average degree . . . 57
- 4.6 Practice problems . . . 59

Bibliography **61**

About this document

Where it's from

If you downloaded this PDF file yourself from https://vertex.degree/contents, presumably you know what you were doing. But in case you're confused or got the PDF file somewhere else, let me explain!

This is not an entire graph theory textbook. It is one part of the textbook *Start Doing Graph Theory* by Mikhail Lavrov, in case it's convenient for you to download a few smaller files instead of one large file. You can find the entire book at https://vertex.degree/; all of it can be downloaded for free.

The complete textbook has some features that the individual parts couldn't possibly have. In this PDF, if you click on a chapter reference from a different part of the book, then you will just be taken to this page, because I can't take you to a page that's not in this PDF file. The hyperlinks in the complete book are fully functional; there is also a preface and an index.

What's inside

In this part of the book, I introduce the fundamental concepts of graph theory that you will need to understand the rest of the textbook. Chapter 1 is an overview of using graphs to solve problems. Chapter 2 covers isomorphic graphs, graph invariants, and subgraphs. Chapter 3 covers walks, paths, cycles, and connected graphs. Chapter 4 covers degrees of vertices and their applications.

The cover

The cover of this PDF shows the volcano graph, with several of its vertices labeled by degree. (The volcano graph was invented by me as an example to use in teaching graph theory, but it is not actually defined in this part of the textbook; it is first used as an example in Chapter 14.)

The license

This document is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License: see https://creativecommons.org/licenses/by-sa/4.0/ for more information.

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.

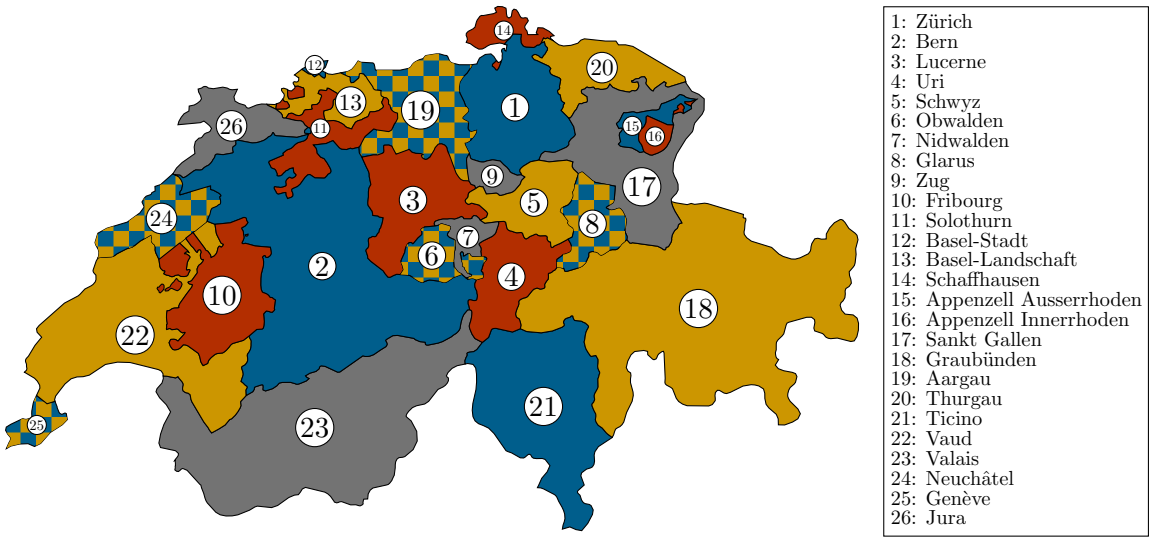


Figure 1.1: A map of Switzerland (coordinates from [6])

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, because it would be very hard to talk about which cantons get which colors, but 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.

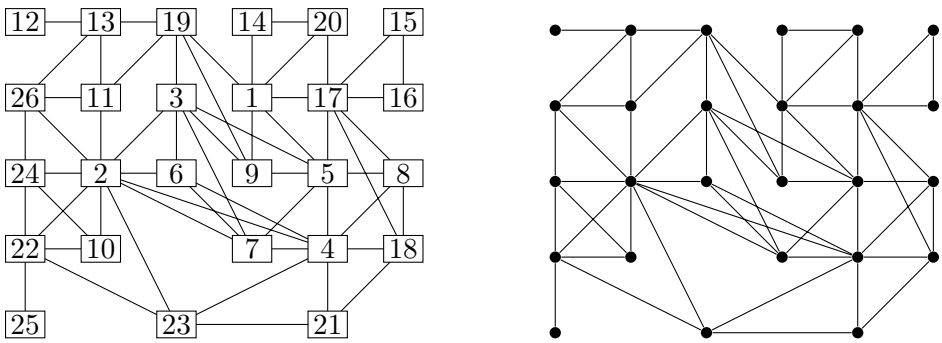


Figure 1.2: Two schematic representations of the Switzerland graph

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):

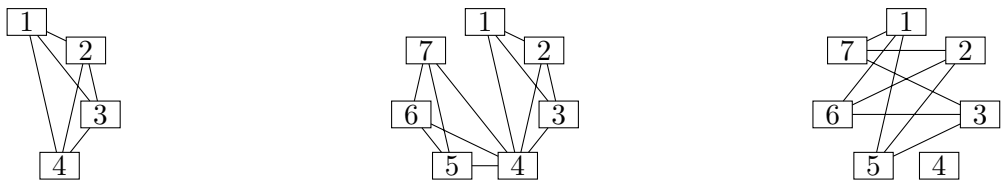
```
(1) -- (5) (1) -- (9) (1) -- (14) (1) -- (17) (1) -- (19) (1) -- (20)
(2) -- (3) (2) -- (4) (2) -- (6) (2) -- (7) (2) -- (10) (2) -- (11)
(2) -- (22) (2) -- (23) (2) -- (24) (2) -- (26) (3) -- (5) (3) -- (6)
(3) -- (7) (3) -- (9) (3) -- (19) (4) -- (5) (4) -- (6) (4) -- (7)
(4) -- (8) (4) -- (18) (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.

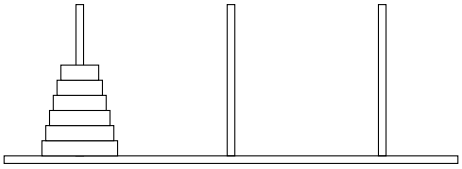


Figure 1.4: The Towers of Hanoi puzzle

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 [1]; 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.

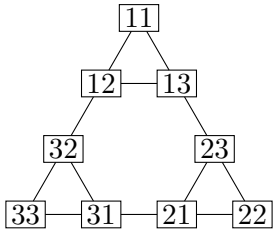


Figure 1.5: The graph representing a 2-disk Towers of Hanoi puzzle

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!

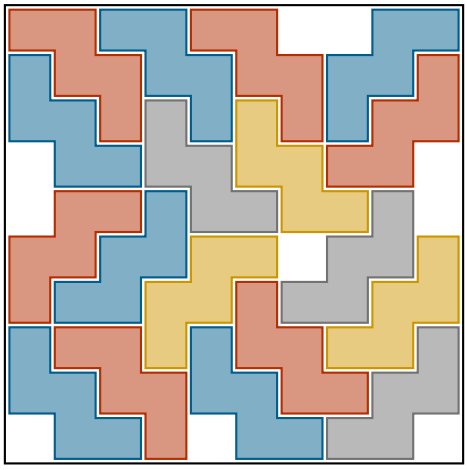


Figure 1.6: A solution to fitting 18 zigzag tiles in a 10×10 grid without overlap

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 [2], 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*.

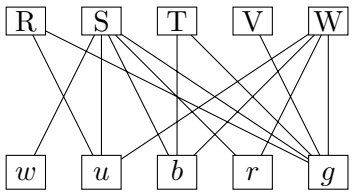
The example of the robot arm is only one instance of this problem appearing in applications: there are many situations that can be modeled by visiting all the vertices of a graph as efficiently as possible!

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 superfically 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

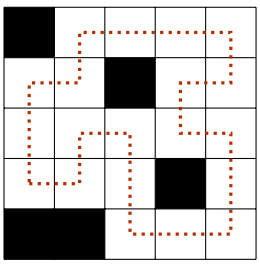
1. Describe how to use a graph to model each of the following puzzles:

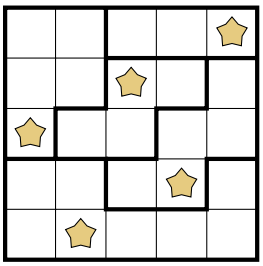
a) A word ladder puzzle, in which one word must be transformed into another by changing one letter at a time. For example, if the goal were to get from "graph" to "Euler", one possible solution might be:

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

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

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.

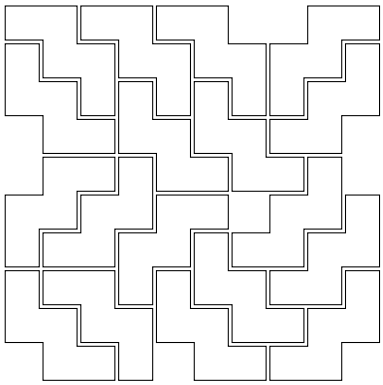
   a) Draw a diagram of the graph where the vertices are ways to place a single 2×2 tile on the grid (there should be 9 vertices), with an edge between two vertices when the tile placements they represent are incompatible.

   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.

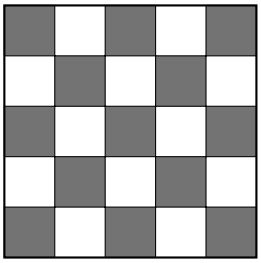
6. This is more of a puzzle than a graph-theoretical question: find a way to fit 12 zigzag-shaped tiles into an 8×8 grid with no overlaps. (How do we know that this is optimal without the use of a computer program?)

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.

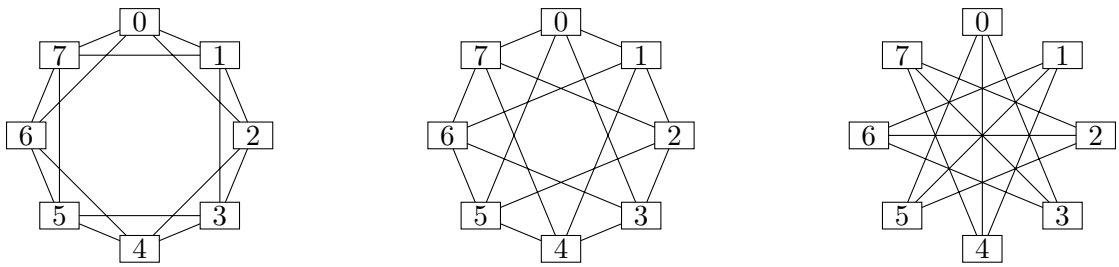
I begin with the definition of circulant graphs for several reasons. They will be useful to us many times later on, because they're a very flexible family of graphs. Their definition is also good to understand now: it will give you some practice with modular arithmetic, which will help us with other 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 $a \bmod n$.*

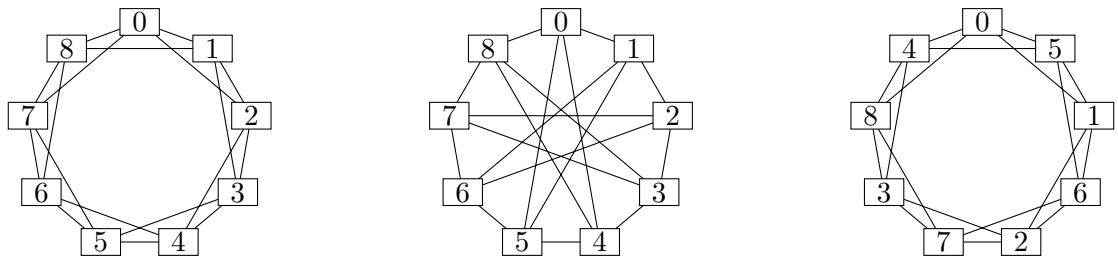
Modular arithmetic is used in situations where numbers "wrap around to the start" after n, so that $n+1$ is the same as 1, and $n+2$ is the same as 2, and so on. A common real-life situation of this type is clock arithmetic: the hours are labeled 1 through 12, but the hour after 12 is 1, not 13. The 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.

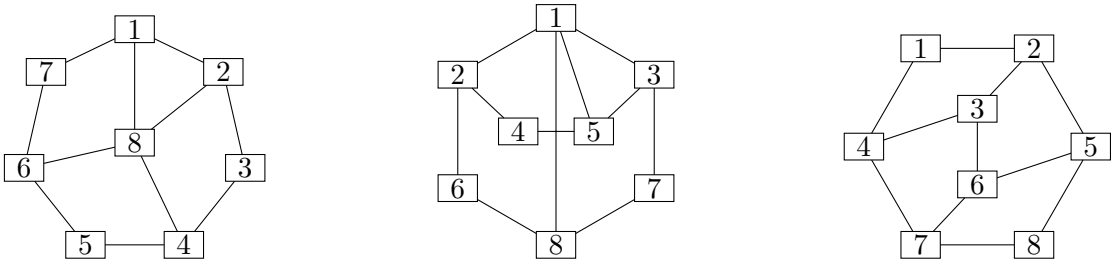
This completes the proof that φ is an isomorphism from $\mathrm{Ci}_n(1, \frac{n-1}{2})$ to $\mathrm{Ci}_n(1,2)$, showing also that the two graphs are isomorphic. □

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)?

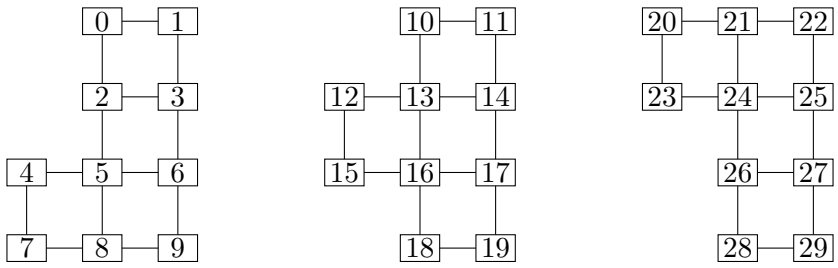
Answer: Yes: if there were an isomorphism $\psi\colon V(G_1) \to V(G_3)$, then $\psi(3)$, $\psi(5)$, and $\psi(7)$ would be three different vertices with two neighbors each. But there are not three such vertices in G_3.

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.

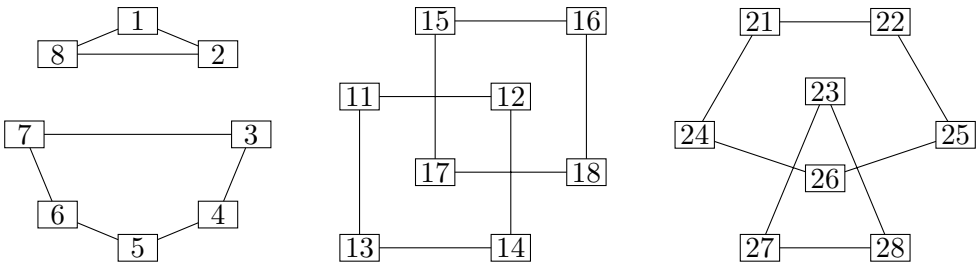
In general, to prove that two graphs are not isomorphic, it's enough to find any difference between the two graphs that any isomorphism must preserve: something that's inherent to the structure of the graph, and not a feature of the vertex labels or the way the graph is drawn.

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.)

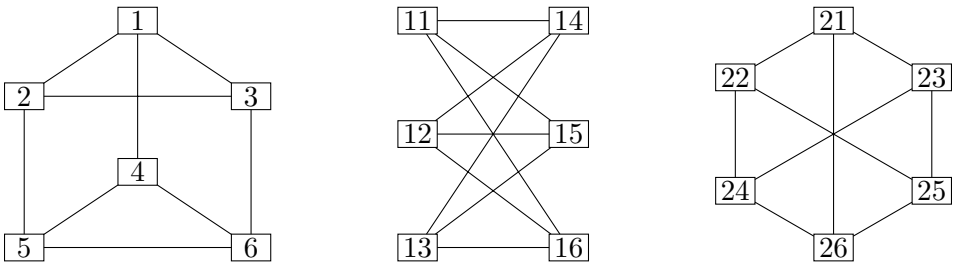
Problem 2.1. *Determine which of the three graphs below is the odd one out, and find an isomorphism between the other two.*



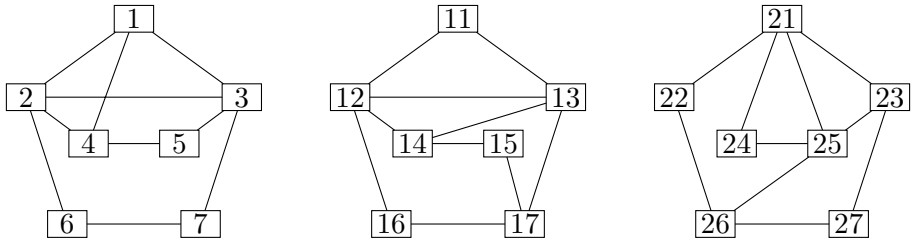
Problem 2.2. *Determine which of the three graphs below is the odd one out, and find an isomorphism between the other two.*



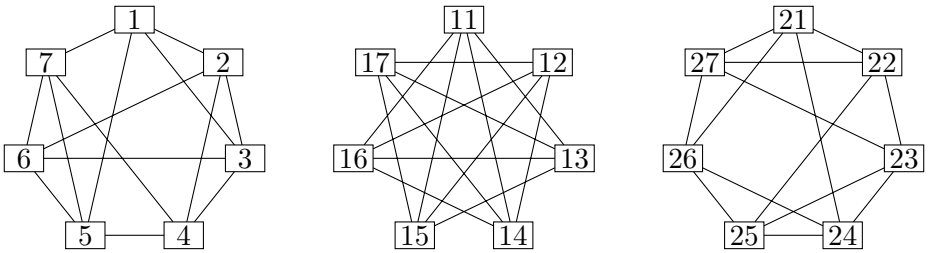
Problem 2.3. *Determine which of the three graphs below is the odd one out, and find an isomorphism between the other two.*



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. $\square$

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.)

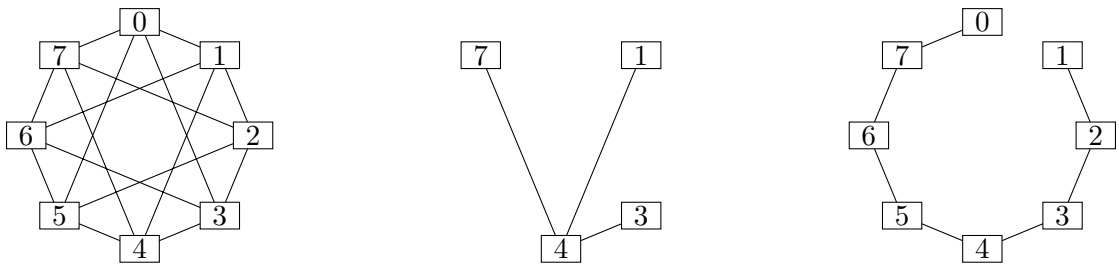
Whether a graph has a "piece" of a certain form is also an invariant; to make this formal, we will need the idea of subgraphs, which are discussed in the final section of this chapter.

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 $V(G) = V(H)$.*

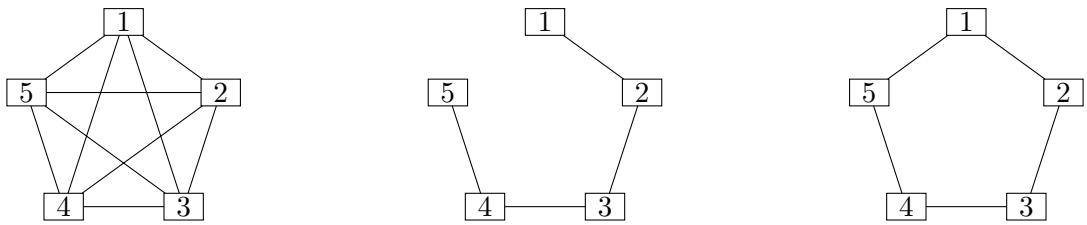
There are no requirements to include any edges: one possible spanning subgraph is the subgraph with all of G's vertices, but no edges at all. Figure 2.4c shows a spanning subgraph of $\mathrm{Ci}_8(1,3)$.

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.

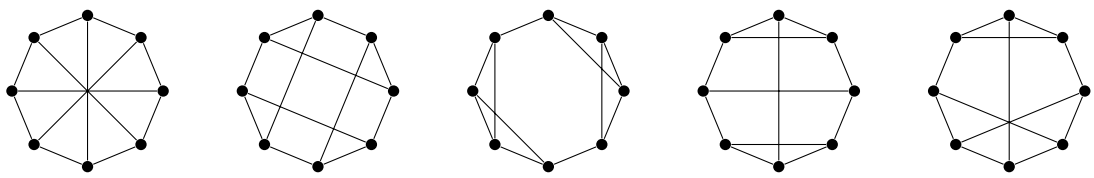
The reason there is no agreement about the vertex sets among graph theorists is because the exact vertex set is hardly ever relevant. If you want to answer questions like, "What is the number of copies of C_4 in a graph G?" then your answer will be the same no matter what names you gave to the vertices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

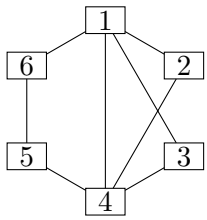
Question:	Is K_n a circulant graph, in this sense?
Answer:	Yes: it is isomorphic to $\mathrm{Ci}_n(1, 2, \ldots, \frac{n-1}{2})$ or $\mathrm{Ci}_n(1, 2, \ldots, \frac{n}{2})$, depending on whether n is odd or even: a circulant graph with all the possible offsets, in either case.

2.6 Practice problems

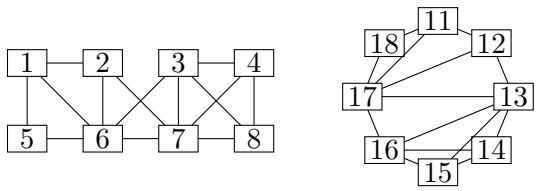
1. Prove that none of these five graphs are isomorphic: find invariants distinguishing them all from each other.



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.

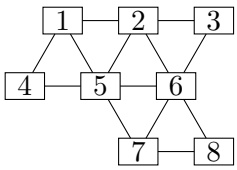
   b) If G contains a copy of K_3, then H contains a copy of K_3.

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.

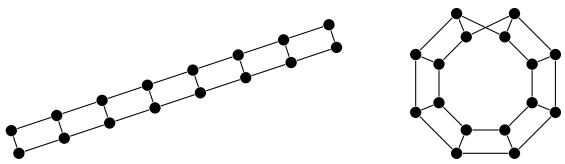
   On the other hand, the converse to (a) is false! Let G be the graph below:



   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.

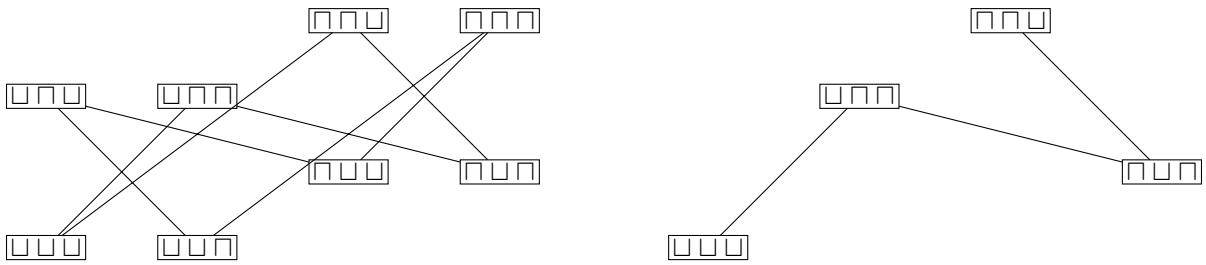
Anyway, justification aside, this also goes to warn you that in graph theory, not every source you read will give you the same definitions. The definitions should be interchangeable, once you learn how. After all, all graph theorists are still doing the same math, no matter how we talk about it.

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* $\boldsymbol{x - y}$ ***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* $\boldsymbol{x - y}$ ***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 $x, y \in V(G)$.*

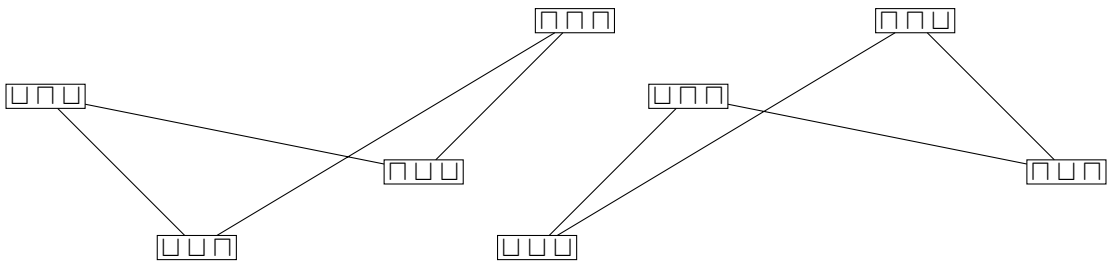


Figure 3.2: The two connected components of the three cup puzzle graph

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 $x \leftrightsquigarrow y$. 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.

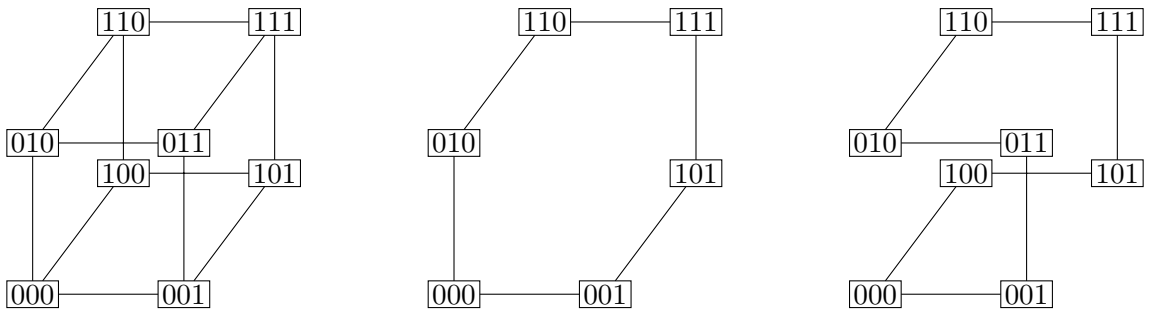
Suppose, for example, that a country's land is separated into two islands, with each island made up of several regions. The two islands will be connected components in a graph that models the country's regions. If we want to find a way to color the map of this country so that adjacent regions get different 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.*

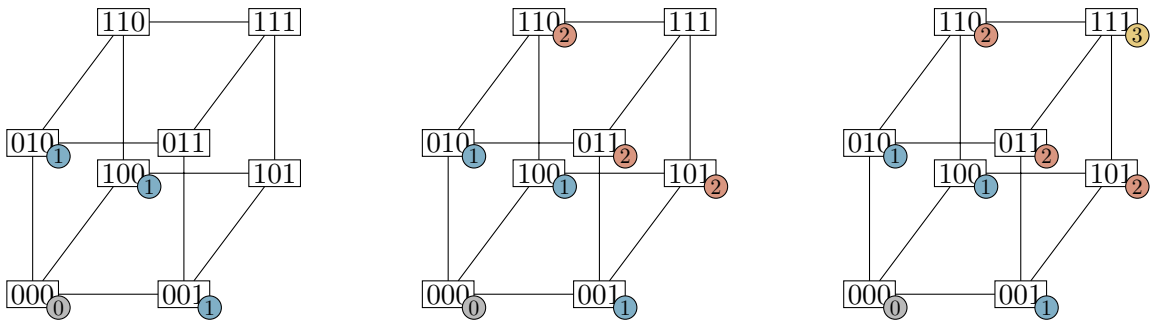
If x and y are in different connected components, then there is no such walk, and we sometimes say that $d(x, y) = \infty$ in that case.

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 (which we called 000 in Figure 3.3a). Take note of the 0 on the bottom left vertex. 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 [5]. 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.

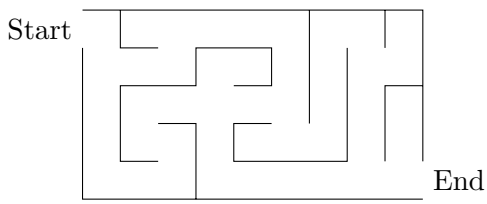
   b) What are the connected components of G?

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

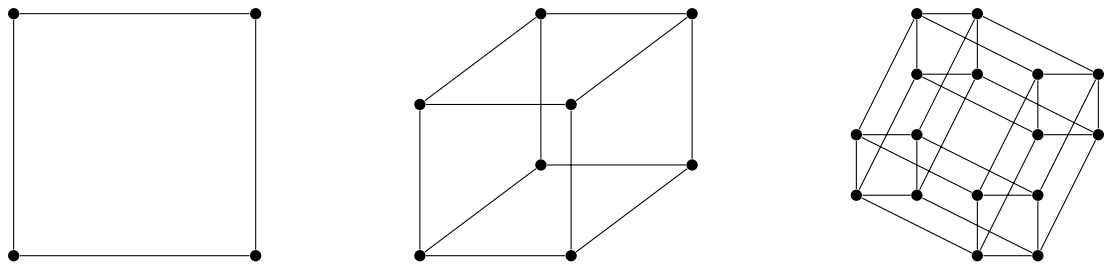
Before we begin discussing vertex degrees, let me introduce a new family of graphs to you: the hypercube graphs. The family of hypercube graphs generalizes the cube graph previously mentioned in Chapter 3:

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 $b_1b_2\dots b_n$. 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.*

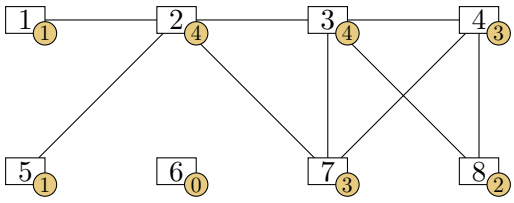


Figure 4.2: An 8-vertex graph labeled with the degrees of its vertices

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 $b_1 b_2 \ldots b_n$ 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.) □

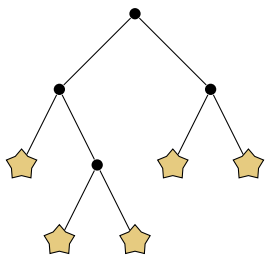


Figure 4.3: An full binary tree with 8 vertices; the marked vertices are leaves.

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

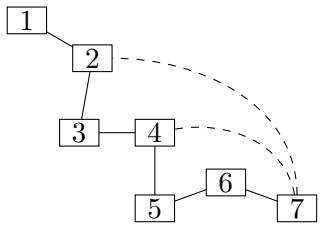
The following theorem, and Corollary 4.7 in the next section, will both be very useful to us many times in future chapters; notably, in Chapter 8 to find cycle decompositions and in Chapter 10 to study graphs with no cycles. In the meantime, they will show a few other ways we can use vertex degrees in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

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

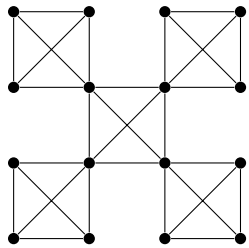
Proof. Let the walk $(x_0, x_1, \ldots, x_l)$ represent a longest path in G.

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 [4] 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 [3] 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.

Bibliography

[1] W. W. Rouse Ball. *Mathematical Recreations and Essays*. Fourth. New York, NY: Macmillan and Company, 1905. URL: https://www.gutenberg.org/ebooks/26839.

[2] 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.

[3] John H. Conway. *Five $1,000 Problems (Update 2017)*. 2017. URL: https://oeis.org/A248380/a248380.pdf (visited on 09/23/2025).

[4] 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.

[5] 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.

[6] SimpleMaps.com. *Free GIS Maps of Switzerland*. Licenced under CC BY 4.0. 2010–2025. URL: https://simplemaps.com/gis/country/ch (visited on 10/26/2025).