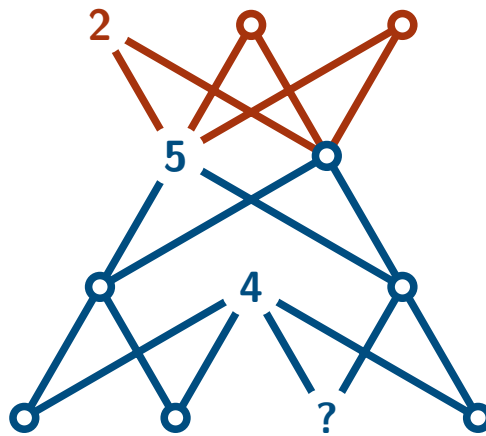


Mikhail Lavrov

Start Doing Graph Theory

Part I: The Beginning



available online at <https://vertex.degree/>

Contents

About this document	3
1 Modeling problems with graphs	4
1.1 Coloring the cantons of Switzerland	4
1.2 The Towers of Hanoi puzzle	7
1.3 A tiling puzzle	10
1.4 Taking photos efficiently	11
1.5 A final puzzle: match the cars to their drivers	13
1.6 Practice problems	14
2 Isomorphisms and subgraphs	17
2.1 A plethora of Wagners	17
2.2 The isomorphism game	21
2.3 Graph properties, or graph invariants	24
2.4 Subgraphs and some common small graphs	27
2.5 Practice problems	29
3 Walks, paths, and cycles	31
3.1 Walks and paths	31
3.2 Connected graphs	33
3.3 Equivalence relations	35
3.4 Closed walks and cycles	37
3.5 Lengths of walks	38
3.6 Distances	40
3.7 Practice problems	41
4 The degree of a vertex	43
4.1 The hypercube graphs	43
4.2 The Handshake Lemma	44
4.3 Degrees and connectedness	48
4.4 Degrees and cycles	49
4.5 Average degree	50
4.6 Practice problems	52
Bibliography	54

About this document

This is Part I of *Start Doing Graph Theory*. It covers what I consider to be the essential background material for understanding the rest of the book, and begins with some motivation for the study of graph theory.

Right now, only parts I through IV are ready. When the book is finished, I plan to provide a variety of ways to read it: a single PDF of the whole book, several smaller PDFs like this one, and eventually an HTML version. For now, think of this document as a preview!

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, just the basic concepts of vertices and edges. If you become curious about the answers to any of these questions—good! However, you will have to wait until a later chapter.

1.1 Coloring the cantons of 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 found the map in Figure 1.1 on Wikipedia [4], and it uses 7 colors to achieve this effect. This is not ideal, practically speaking, for a couple of reasons:

- The more colors you use, the harder it is to choose a colorblind-friendly palette of colors.
- Switzerland has many lakes, which are light blue in Figure 1.1. So it would be better if we had fewer colors and did not have to use a shade of light blue for any of the cantons.

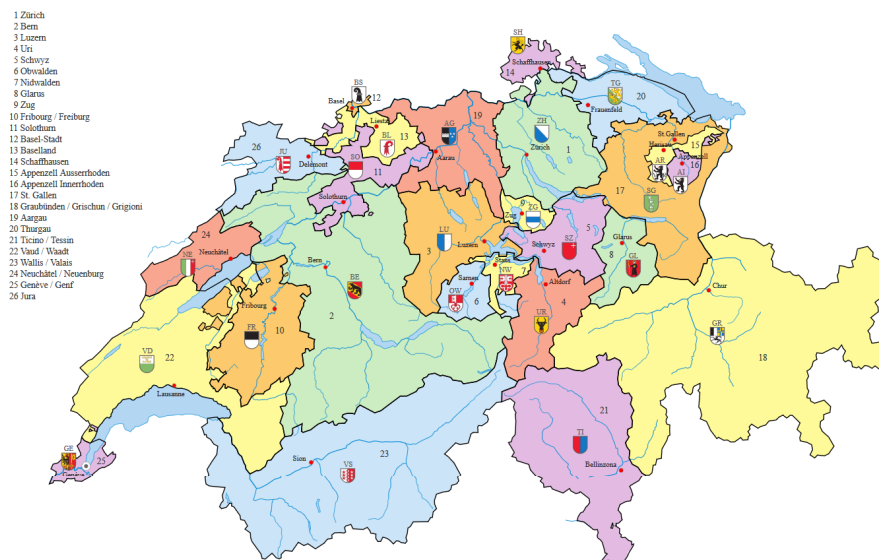


Figure 1.1: A map of the cantons of Switzerland

How many colors are necessary to color the cantons of Switzerland?¹ 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, St. Gallen, and Zürich: 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

¹If 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.

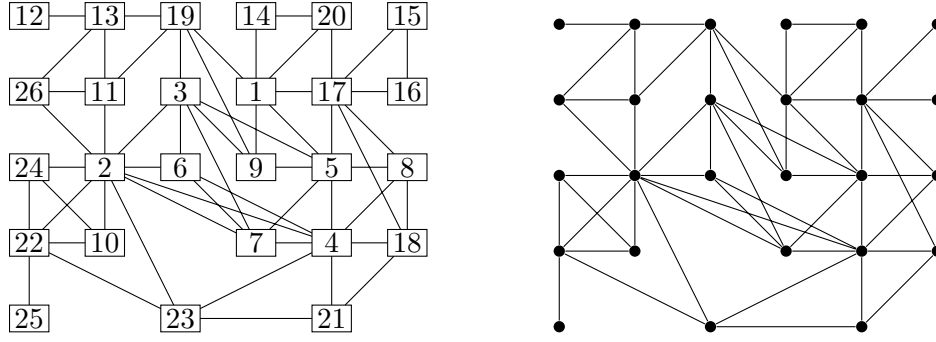


Figure 1.2: Two schematic representations of the Switzerland graph

geographic location. This is just as good for our purposes; maybe even better, because it does not include any distracting details!

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);
```

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.

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 .

The philosophy behind this definition is that the edges represent some kind of symmetric relationship between whatever kind of objects the vertices are: in our case, sharing a border. Situations like “sharing a border” are a common way to obtain a graph, and a lot of secondary terminology in graph theory reflects this sort of application:

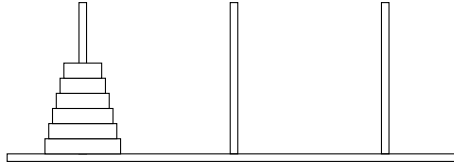


Figure 1.3: The Towers of Hanoi puzzle

Definition 1.2. If a graph G contains vertices x and y and an edge $\{x, y\}$, then we say that x and y are **adjacent**, or that x is a **neighbor** of y (and vice versa).

Definition 1.3. The edge $\{x, y\}$ itself, in that case, might be said to go **between** x and y , or to **join** x and y .²

Definition 1.4. We call x and y the **endpoints** of the edge $\{x, y\}$. For a slightly fancier option, we might say that x and y are **incident on** the edge $\{x, y\}$, or that edge $\{x, y\}$ is **incident on** x and y .

From now on, whenever possible, we will write the edge $\{x, y\}$ simply as xy , only using set notation when it's necessary to avoid ambiguity. For example, we would not want to represent an edge in the Switzerland graph as “117”, because it's not clear if this means $\{1, 17\}$ or $\{11, 7\}$.

Later on in this book, we will define a **coloring** of a graph to be a function from V 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, $f(20) = \text{blue}$ 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.

One final note: according to some people, what we've defined here is not a fully general graph but a **simple graph**. What those people call a graph, I will call a **multigraph**; I will define this term in Chapter 4.

1.2 The Towers of Hanoi puzzle

In the Towers of Hanoi 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 1.3.

In this puzzle, 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 counterintuitive 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

²I would avoid using the words “connect” or “connected” in this context; they are somewhat overloaded in graph theory.

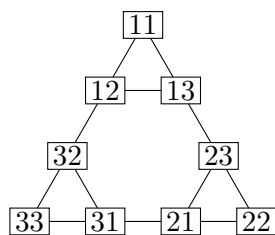


Figure 1.4: The graph representing a 2-disk Towers of Hanoi puzzle

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 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 1.3 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 1.4, 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 1.4, 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

$$x_0, x_1, x_2, \dots, x_{m-1}, x_m$$

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 $i = 1, \dots, m$, 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. 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 1.4 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

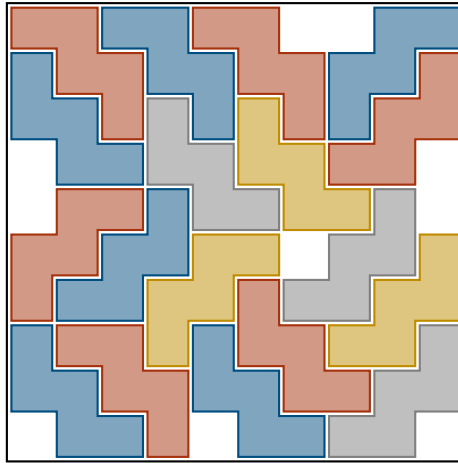



Figure 1.5: A solution to fitting 18 zigzag tiles in a 10×10 grid without overlap


yet. However, in that bigger graph, it would solve essentially the same problem of finding a minimum-length walk!

1.3 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 1.5.

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 17.

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. Reversing the rule for adjacencies like this gives the **complement** of the original graph.

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. On my laptop, it took 48 seconds to find the largest independent set in the graph: longer than it took me to describe the problem to my computer!

1.4 Taking photos efficiently

Here is an application of graph theory to engineering.³ 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!

³I found it in a 2022 paper by Bottin, Boschetti, and Rosati [1], 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.

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**. 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 path.) 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 is the least number possible. Such a walk has a special name: it is a **Hamilton path** in the graph. Hamilton paths are discussed in Chapter 16 of this book.

The very first time an instance of this problem was considered was as a mathematical puzzle! In chess, the “knight 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 1.6a: 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 1.6b, is precisely a knight tour of the chessboard.

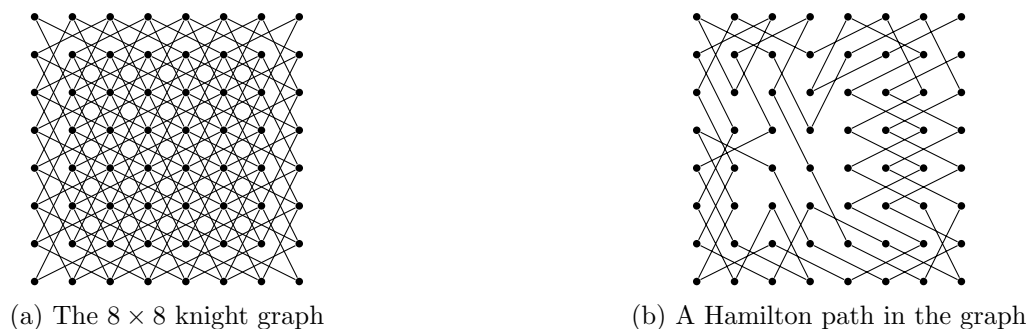


Figure 1.6: The knight tour problem

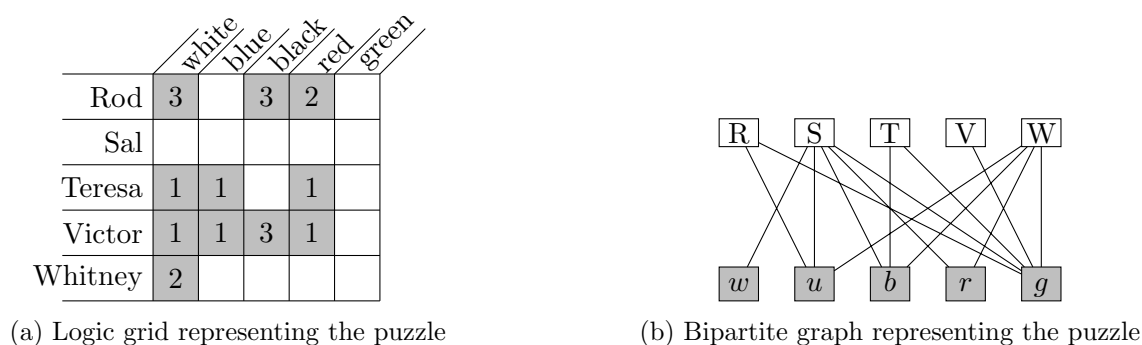


Figure 1.7: Two ways to think about matching cars to people

1.5 A final puzzle: match the cars to their 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 1.7a, 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 1.7b 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 and shaded 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 12.

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 1.7b is not necessarily the best visualization if you want to try to solve the logic puzzle. However, thinking about the problem as a graph is rewarding for two reasons:

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

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

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

Question: Suppose a graph has $n = 99$ vertices. Then a perfect matching should have $n/2$ edges, which is 49.5. How can we make sense of this?

Answer: In a graph with 99 vertices (or any other odd number), a perfect matching cannot exist—so it's okay that our formula gives a nonsense answer. If we have 99 objects, we cannot divide them into pairs; one object will be left out at the end.

1.6 Practice problems

1. 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, of

- a) The 7 regions, if you just want a bit of practice.
 - b) The 19 counties and Budapest, if you want to draw a bigger graph.
2. 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.
 3. 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 1.4 is a diagram of H_2 .

- a) Draw only one part of H_3 : the part containing the 9 vertices

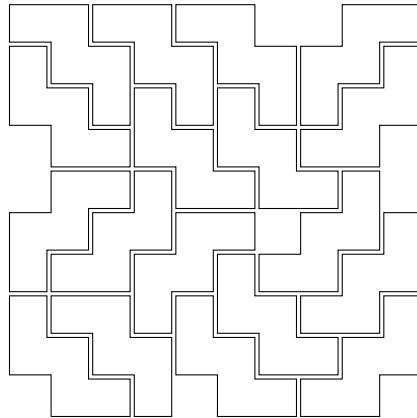
$$\{111, 112, 113, 121, 122, 123, 131, 132, 133\}.$$

- 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 ?
4. 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.

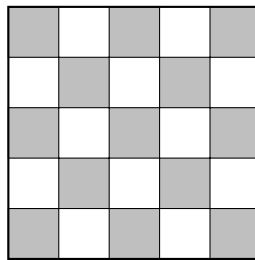
- a) Draw a diagram of the graph where the vertices are ways to place a single 2×2 tile on the grid (there should be 9 vertices), with an edge between two vertices when the tile placements they represent are incompatible.
 - b) Find an independent set in the graph representing the “boring” solution, 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.
5. An **interval graph** is a graph whose vertices are intervals of the real line, with an edge between two vertices when the two intervals overlap.
 - a) Draw a diagram of the interval graph with vertices $[8, 13]$, $[9, 11]$, $[10, 14]$, $[12, 15]$, $[16, 18]$, $[17, 19]$.
 - b) Suppose that the intervals represent times of day that certain events are happening (for example, TV shows you want to watch). What would an independent set in this graph correspond to?
 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 this tiling 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. I like to describe this as a jump to another location exactly $\sqrt{5}$ units away. (Some knight moves are illustrated in Figure 1.6b.)

- a) (A puzzle) Find a knight tour of the 5×5 chessboard, shown below (a Hamilton path in the 5×5 knight graph).



- b) (A math problem) Prove that every knight tour that can be found as an answer to part (a) must begin and end on one of the dark-colored squares.

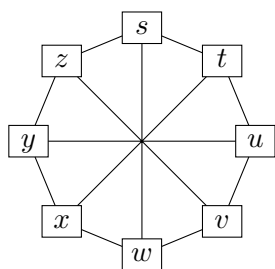
2 Isomorphisms and subgraphs

The purpose of this chapter

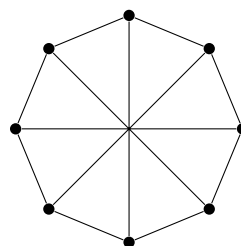
The definition of a graph isomorphism is sufficiently fundamental that no self-respecting graph theory book could skip it. Most authors wouldn't put this chapter so close to the beginning of the book, though.

The reason I wanted to introduce isomorphic graphs early on is for the sake of the isomorphism game in Section 2.2. 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 isomorphism invariants.

2.1 A plethora of Wagners



(a) A labeled diagram of the Wagner graph



(b) An unlabeled diagram of the Wagner graph

Figure 2.1: Two drawings of the Wagner graph

Let's begin by looking a picture; specifically, Figure 2.1, which shows two diagrams of a graph known as the **Wagner graph**. (The Wagner graph is a small graph with many interesting applications, which is named after graph theorist Klaus Wagner; we will see it again later in this book.) What's going on in this picture?

The answer is simple: Figure 2.1a is a labeled diagram, and Figure 2.1b is unlabeled. Perhaps today we would like to call the Wagner graph G , and we'd like it to have vertex set $V(G) = \{s, t, u, v, w, x, y, z\}$ and edge set $E(G) = \{st, sw, sz, tu, tx, uv, uy, vw, vz, wx, xy, yz\}$. It would be fair to say that *both* of the diagrams in Figure 2.1 are diagrams of this graph G ; it's simply that in Figure 2.1a, we chose to label the vertices, and in Figure 2.1b, we did not. This is something we'll often do when the labels don't matter. If the labels did matter, then Figure 2.1b would be a bit ambiguous, which brings us to the next example...

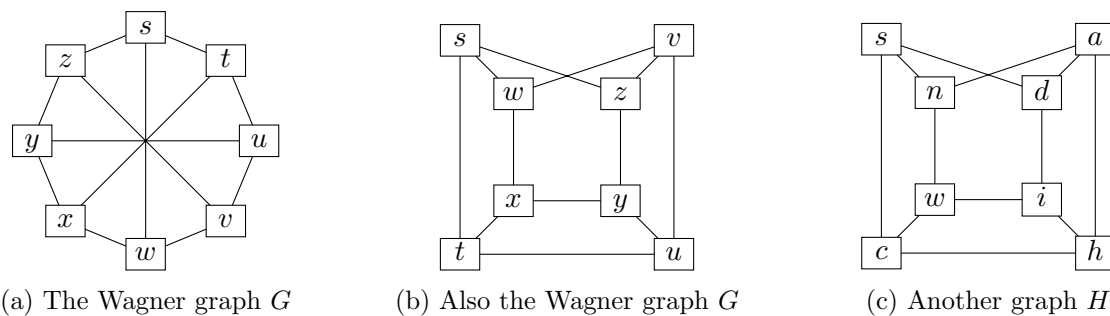


Figure 2.2: The Wagner graph... and a friend?

Figure 2.2 shows three diagrams. The first, Figure 2.2a, shows the very same diagram of the Wagner graph G again, for reference. However, Figure 2.2b is also a diagram of G , even though it looks different! The graph in the diagram has the same 8 vertices $\{s, t, u, v, w, x, y, z\}$ and the same 12 edges $\{st, sw, sz, tv, tx, uv, uy, vw, vz, wx, xy, yz\}$. According to our definition, that's all there is to a graph. It's irrelevant that the vertices have been moved around so that the shape is now unrecognizable.

Meanwhile, the graph in Figure 2.2c, which we'll call H , is clearly different. Its vertex set is $V(H) = \{s, a, n, d, w, i, c, h\}$, which is not the same set as $V(G)$, so it feels unnecessary to even mention that the edge set $E(H) = \{ad, ah, an, ch, cs, cw, di, ds, hi, hw, ns, nw\}$ is not the same set as $E(G)$. And yet, Figure 2.2b is almost identical to Figure 2.2c: they have vertices and edges in all the same places, with only the labels changed! If you think about the sort of questions we asked in Chapter 1—for example, the largest number of vertices we can select without picking two adjacent vertices—it's clear that any answer we find in G will also work in H , and vice versa.

Question: In the graph G , the set $\{t, w, y\}$ is independent: no two vertices in this set are adjacent. Does this correspond in any way to an independent set in H ?

Answer: Yes, but we have to do some “translation”. The vertices t, w, y are in the same place in Figure 2.2b as the vertices c, n, i in Figure 2.2c, so $\{c, n, i\}$ is an independent set in H .

We say that graph H is “isomorphic” to graph G : by changing the names of the vertices of G , we can turn it into H . If we had to describe *how* G corresponds to H without drawing the diagram, we'd want to list out the correspondences between the vertices: vertex s in G corresponds to vertex s in H , vertex t in G corresponds to vertex c in H , and so on. The mathematical object that describes such a correspondence is a function: a function $\varphi: V(G) \rightarrow V(H)$. Not just any function will do, however:

Definition 2.1. A function $\varphi: V(G) \rightarrow V(H)$ is an **isomorphism from G to H** if it 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 $\varphi^{-1}: V(H) \rightarrow V(G)$.

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.

Further, we say that **G and H are isomorphic** or **G is isomorphic to H** if there exists an isomorphism from G to H .

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 2.2b and Figure 2.2c are the same, the two properties are just formally describing what you'd do.¹ 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 on all of $V(G)$, 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 $xy \in E(G)$ if and only if $\varphi(x)\varphi(y) \in E(H)$).

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

vertex of G	s	t	u	v	w	x	y	z
$\varphi(\text{vertex})$	s	c	h	a	n	w	i	d

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 G to names in H . In the “language” of G , a sentence like “ s is adjacent to t , but not to u ” is true. The same sentence is true in H , but needs translation first! When translated, it becomes “ $\varphi(s)$ is adjacent to $\varphi(t)$, but not to $\varphi(u)$ ”, or “ s is adjacent to c , but not to h ”.

(Just as with foreign languages, such a dictionary can have cognates—vertices like s that have the same role in both graphs. It can also have false cognates: both graphs have a vertex w , but G 's w does not correspond to H 's w .)

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 even integers $n = 2k$, the following two graphs are isomorphic:*

- (i) *The graph M_n with $V(M_n) = \{x_1, x_2, \dots, x_n\}$ and $E(M_n) = \{x_i x_{i+1} : 1 \leq i \leq n-1\} \cup \{x_n x_1\} \cup \{x_i x_{i+k} : 1 \leq i \leq k\}$.*
- (ii) *The graph M'_n with $V(M'_n) = \{y_1, \dots, y_k\} \cup \{z_1, \dots, z_k\}$ and $E(M'_n) = \{y_i z_i : 1 \leq i \leq k\} \cup \{y_i y_{i+1} : 1 \leq i \leq k-1\} \cup \{z_i z_{i+1} : 1 \leq i \leq k-1\} \cup \{y_k z_1, z_k y_1\}$.*

¹With the diagram, though, there's a handy trick. While looking at Figure 2.2, cross your eyes so that Figure 2.2b and Figure 2.2c are superimposed in your vision. Technically, what happens is that your left eye is looking at Figure 2.2c and your right eye is looking at Figure 2.2b. You should see a single graph, with only some of the letters shimmering in your vision because they differ between the diagrams. If the two diagrams disagree in a vertex or edge, that vertex or edge will also shimmer in your vision, and you'll spot it instantly!

Question: What is the connection between this proposition and the diagrams in Figure 2.2?

Answer: When $n = 8$, both M_8 and M'_8 are isomorphic to the Wagner graph! Moreover, if we draw M_8 by putting x_1, \dots, x_8 in a circle, we get a diagram that looks like Figure 2.2a. If we draw M'_8 by putting y_1, \dots, y_4 in a big square and z_1, \dots, z_4 in a smaller square inside it, we get a diagram that looks like Figure 2.2b or Figure 2.2c.

Proof. We will prove that the following function $\varphi: V(M_n) \rightarrow V(M'_n)$ is an isomorphism from M_n to M'_n :

$$\varphi(x_i) = \begin{cases} y_i & 1 \leq i \leq k \\ z_{i-k} & k+1 \leq i \leq n. \end{cases}$$

This is a bijection, because it has an inverse: $\varphi^{-1}(y_i) = x_i$ and $\varphi^{-1}(z_i) = x_{i+k}$ for $k = 1, \dots, n$.

Next, we check that φ maps vertices adjacent in M_n to vertices adjacent in M'_n . There are several cases to check, because we defined $E(M_n)$ as the union of three sets, and also because φ has a piecewise definition.

1. For an edge of the form $x_i x_{i+1}$ with $1 \leq i \leq k-1$, $\varphi(x_i) = y_i$ and $\varphi(x_{i+1}) = y_{i+1}$, and all pairs $y_i y_{i+1}$ are edges of M'_n . Similarly, for an edge of the form $x_i x_{i+1}$ with $k \leq i \leq n-1$, $\varphi(x_i) = z_{i-k}$ and $\varphi(x_{i+1}) = z_{i+1-k}$, which are z_j and z_{j+1} for $j = i-k$. All pairs $z_j z_{j+1}$ are edges of M'_n for $1 \leq j \leq k-1$.
2. For the edge $x_k x_{k+1}$, we have $\varphi(x_k) = y_k$ and $\varphi(x_{k+1}) = z_1$, and we specifically included $y_k z_1$ as an edge of M'_n . Similarly, for the edge $x_n x_1$, we have $\varphi(x_n) = \varphi(x_{2k}) = z_k$ and $\varphi(x_1) = y_1$, and we specifically included $z_k y_1$ as an edge of M'_n .
3. For an edge of the form $x_i x_{i+k}$ with $1 \leq i \leq k$, $\varphi(x_i) = y_i$ and $\varphi(x_{i+k}) = z_i$, and all pairs $y_i z_i$ are edges of M'_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 M_n to non-adjacent vertices in M'_n , or that φ^{-1} sends adjacent vertices in M_n to adjacent vertices in M'_n . In this case, however, we can skip that step by doing some counting instead.

From the definitions, $|E(M_n)| = (n-1) + 1 + k = 3k$ and $|E(M'_n)| = k + (k-1) + (k-1) + 2 = 3k$: both graphs have the same number of edges. We already know that φ is a bijection that sends each of the $3k$ edges of M_n to an edge of M'_n . This “uses up” all $3k$ edges of M'_n . So for a pair of vertices that are not adjacent in M_n , their images cannot be adjacent in M'_n , verifying the converse.

This completes the proof that φ is an isomorphism from M_n to M'_n , showing also that the two graphs are isomorphic. \square

A special kind of isomorphism is a **automorphism**: an isomorphism from a graph G to itself. That definition doesn't seem useful; we already know that G is the same as G , so why do we need an automorphism? In fact, every graph G always has one automorphism, the **identity** or **trivial** automorphism: the function $\varphi: V(G) \rightarrow V(G)$ given by $\varphi(x) = x$ for all $x \in V(G)$.

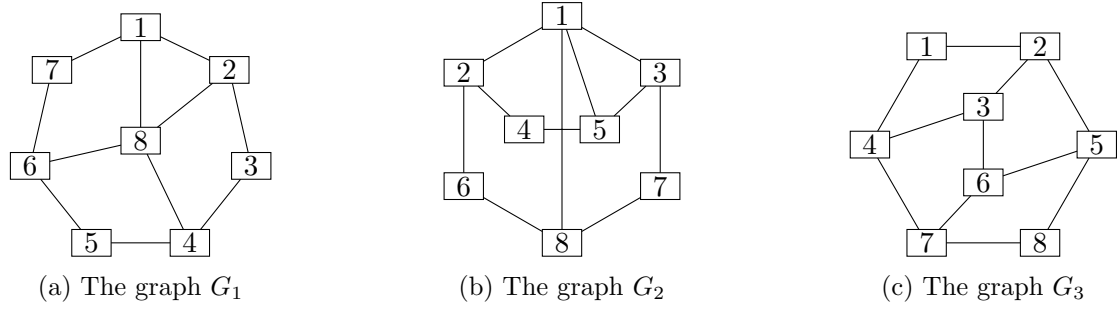


Figure 2.3: A three-graph set for the isomorphism game

The reason automorphisms are studied is that the non-identity automorphisms reveal symmetries of the graph. For example, looking at Figure 2.2b, we see that the Wagner graph G has a vertical line of symmetry: it looks exactly like its own mirror image if you swap left and right. The formal way to state this, independent of the diagram we draw, is that the function α swapping left and right in the diagram is an automorphism: the function given by

vertex of G	s	t	u	v	w	x	y	z
$\alpha(\text{vertex})$	v	u	t	s	z	y	x	w

In Figure 2.2a, we see a different kind of symmetry: the shape of the diagram is unchanged if it is rotated 45° . This corresponds to the automorphism β given below:

vertex of G	s	t	u	v	w	x	y	z
$\beta(\text{vertex})$	t	u	v	w	x	y	z	s

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

2.2 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 $\varphi: V(G_1) \rightarrow V(G_2)$ 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 $\varphi(6)$. 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
$\varphi(\text{vertex})$	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 $\psi: V(G_1) \rightarrow V(G_3)$ is an isomorphism, then $\psi(8)$ must be a vertex of G_3 with four neighbors: $\psi(1)$, $\psi(2)$, $\psi(4)$, and $\psi(6)$. 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 $\psi: V(G_1) \rightarrow V(G_3)$, 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 .

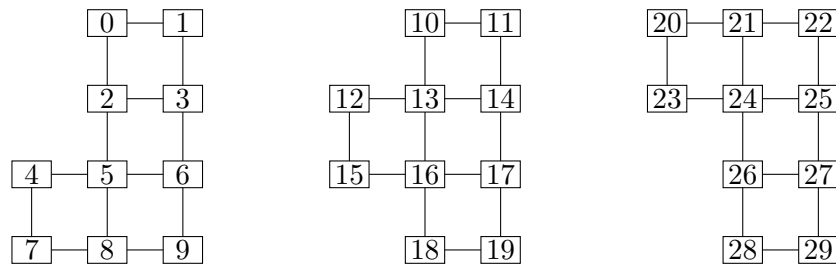
Question: Is it correct to say that G_1 is isomorphic to G_2 because for every number k , the two graphs have the same number of vertices with k neighbors?

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.

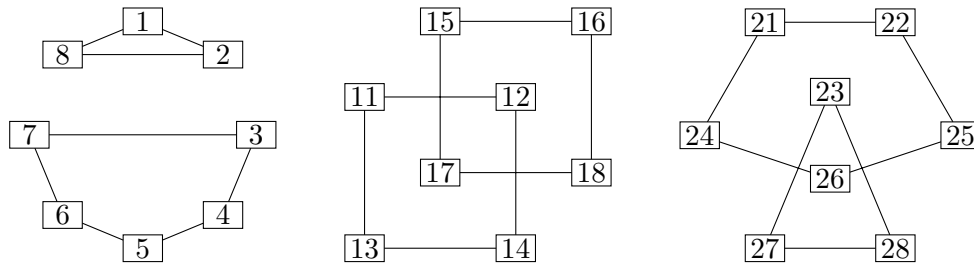
In general, to prove that two graphs are not isomorphic, it's enough to find any difference between the two graphs that any isomorphism must preserve: something that's inherent to the structure of the graph, and not a feature of the vertex labels or the way the graph is drawn.

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.)

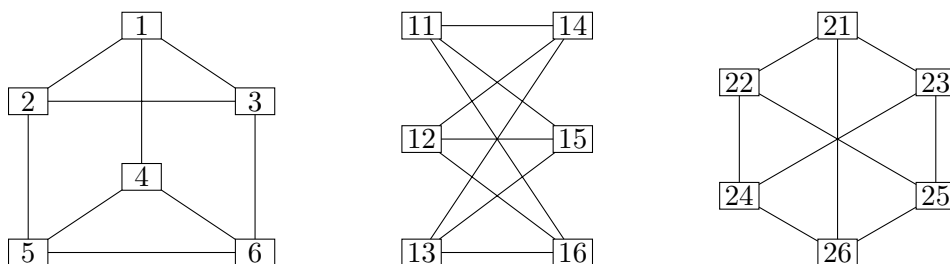
Problem 2.1. Determine which of the three graphs below is the odd one out, and find an isomorphism between the other two.



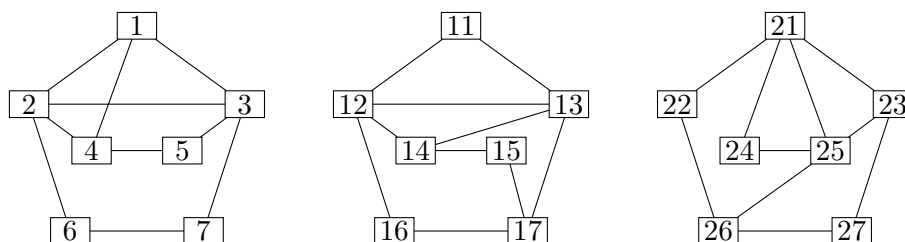
Problem 2.2. Determine which of the three graphs below is the odd one out, and find an isomorphism between the other two.



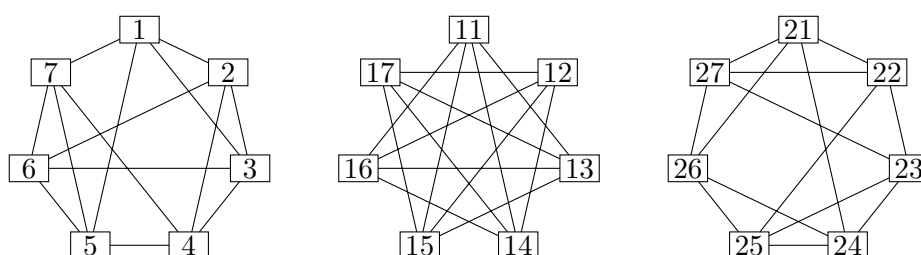
Problem 2.3. Determine which of the three graphs below is the odd one out, and find an isomorphism between the other two.



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.



2.3 Graph properties, or 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 either called a “graph property” or a “graph invariant”:

Definition 2.2. A property of a graph (that is, a result of any kind that can be computed from the graph) is called a **graph property** or **graph invariant** if, 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 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.

There is no difference between saying “graph invariant” or “graph property”, but using the word “graph property” is also taking a philosophical stance. The implication when we say that is 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 property: we have to prove it. I will show you two of these proofs, one for a numerical property and one for a true/false property. 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 properties.*²

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 $\varphi: V(G) \rightarrow V(H)$, 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 $E(G) \rightarrow E(H)$, 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 $\varphi^{-1}: V(H) \rightarrow V(G)$. 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 property. \square

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. *Having a vertex adjacent to all other vertices is a graph invariant.*

²Some 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 try to stay consistent with this terminology, but it feels very unintuitive to me, so I won't rely on it to communicate.

Proof. Let G and H be two isomorphic graphs, and let $\varphi: V(G) \rightarrow V(H)$ 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 $\varphi^{-1}: V(H) \rightarrow V(G)$. 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 property; if it's not, the number of **connected components** and how big they are are graph properties. (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 on it.

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**.

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.

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 isomorphism. The other exception is that graph theorists often study structures which are graphs plus some extra data: an example mentioned in Chapter 1 is weighted graphs, with a number attached to every edge. When these come into play, sometimes we stop looking at arbitrary graph isomorphisms and start looking only at graph isomorphisms that “play nicely” with the extra data we're considering. For example, a weighted graph isomorphism from a weighted graph G to a weighted graph H is a graph isomorphism φ with the additional property that for all edges $xy \in E(G)$, the weight of xy

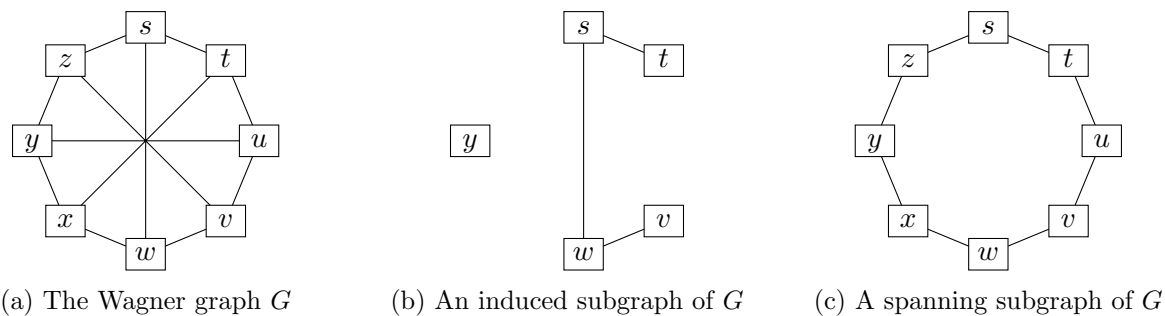


Figure 2.4: The Wagner graph and some of its subgraphs

in G is the same as the weight of $\varphi(x)\varphi(y)$ in H . If we restrict the isomorphisms we allow in this way, then this expands the set of properties that no isomorphism can change, giving us additional invariants.

2.4 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 Wagner graph—again!—and some of its subgraphs.

Definition 2.3. If G is a graph, a **subgraph of G** is simply a graph H 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 ; that edge just wouldn't make sense.

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

Definition 2.4. If G is a graph, a **spanning subgraph of G** is a subgraph H with $V(G) = V(H)$.

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 the Wagner graph.

Definition 2.5. If G is a graph, an **induced subgraph of G** is a subgraph that includes all the edges it possibly could: all the edges $xy \in E(G)$ such that $x \in V(H)$ and $y \in V(H)$. For a subset $S \subseteq V(G)$, we write $G[S]$ for the **subgraph of G induced by S** : the unique induced subgraph of G with vertex set S .

Figure 2.4b shows an induced subgraph of the Wagner graph: the subgraph induced by the set $\{s, t, v, w, y\}$.

It's also common to define a subgraph that contains almost of G by deleting vertices or edges. There are two simple cases: if $xy \in E(G)$, then we write $G - xy$ for the subgraph including

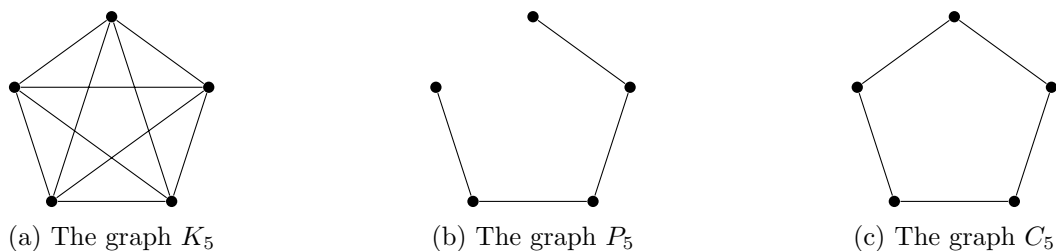


Figure 2.5: Several common graphs

all vertices of G and all edges except edge xy , and 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 on 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 are deleted. For example, Figure 2.4b could also be described as $G - \{u, x, z\}$, where G is the Wagner graph.

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):

Definition 2.6. 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 property “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 2.7. For any $n \geq 1$, the **complete graph K_n** is the graph with vertex set $\{1, \dots, n\}$ and all $\binom{n}{2}$ possible edges $\{i, j\}$ where $1 \leq i < j \leq n$. A copy of a complete graph is often referred to as a **clique**.

Definition 2.8. For any $n \geq 1$, the **path graph P_n** is the graph with vertex set $\{1, \dots, n\}$ and $n - 1$ edges: the edges $\{i, i + 1\}$ where $1 \leq i \leq n - 1$.

Definition 2.9. 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 $\{1, n\}$.

Figure 2.5 includes an example of each of these families, but I have drawn unlabeled diagrams of each of them. That’s because the exact vertex set is hardly ever relevant, and may also be different if you consult other sources. If you want to answer questions like, “What is the number of copies of C_4 in the Wagner graph?” then the exact set $V(C_4)$ is irrelevant.

Really, we would like to say that C_4 is an “unlabeled graph” whose vertices don’t have names. We can’t really do that: if the vertices of a graph didn’t have names, we wouldn’t be able to say which edges it has. However, the idea of unlabeled graphs is a useful fiction in counting problems. We say that the **number of unlabeled graphs** of a certain type is the largest

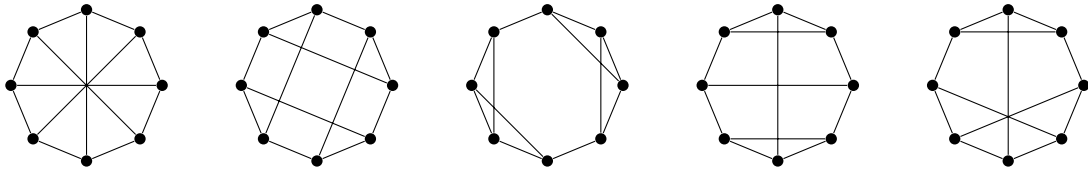
possible number of non-isomorphic graphs of that type. For example, there are 11 different unlabeled graphs on 4 vertices. On the other hand, it is nonsense to ask, “How many graphs on 4 vertices are there?” without the qualifier: there are infinitely many, because there are infinitely many possible 4-element sets that could be the vertex set.

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 “edge” $\{1, 1\}$, and when $n = 2$, the edge $\{1, 2\}$ would already be present in P_2 . Later on, though, we will encounter **multigraphs** that could be called C_1 and C_2 .

2.5 Practice problems

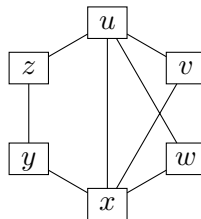
1. Prove that none of these five graphs are isomorphic: find invariants distinguishing them all from each other.



2. Find all 11 different unlabeled graphs on 4 vertices.
3. How many copies of C_4 are there in the Wagner graph? How many copies of C_5 are there?
4. Find all automorphisms of the graph shown below: that is, all functions

$$\varphi: \{u, v, w, x, y, z\} \rightarrow \{u, v, w, x, y, z\}$$

that preserve the edges of the graph.

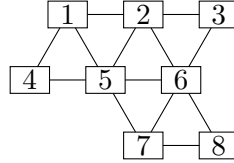


(Hint: there are four, and one of them is very boring.)

5.
 - 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 .
6. 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 .
7. Define two vertices x, y of a graph G to be **similar** if there is an automorphism of G that takes x to y .
- a) Prove that if x and y are similar, then $G - x$ and $G - y$ are isomorphic. (*Earlier in the chapter, we defined $G - x$ to be the subgraph induced by the set $V(G) \setminus \{x\}$: the graph obtained from G by deleting x and all edges incident on x .*)

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.)

starting state, a single move can take us to the $\cap\cap\cup$ state if the first two cups are flipped, or to the $\cup\cup\cap$ state if the last two cups are flipped. The goal of the puzzle is to flip all three cups upside down.¹

Figure 3.1 shows a graph of all the states possible in the three cups 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.) A possible “history” of your victim’s fruitless attempts to solve the puzzle is a sequence of vertices in which any two consecutive vertices are adjacent in the graph: a sequence such as for example

$$\cup\cup\cup, \cup\cap\cap, \cap\cup\cap, \cap\cap\cup, \cap\cup\cap.$$

Generalizing this idea, we make the following definition:

Definition 3.1. A **walk** in a graph G is a sequence $x_0, x_1, x_2, \dots, x_k$ of vertices of G , such that for each $i = 1, \dots, k$, 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 called an $x - y$ walk.

Provided the condition defining a walk is satisfied, we can repeat vertices, and even walk back and forth along the same edge many times if we so choose. It will later be useful to know that a sequence of only a single vertex x is an $x - x$ walk. (The condition that consecutive vertices are adjacent is satisfied trivially, meaning that there is nothing to check.)

In the three cups puzzle, we are looking for a $\cup\cup\cup - \cap\cap\cap$ walk, but if you think about it, the freedom to revisit vertices isn’t really necessary.

Question: Suppose there were a solution to the three cups puzzle that passed through the $\cap\cup\cap$ state multiple times. How could we get a shorter solution?

Answer: We could skip all the steps between the first visit to $\cap\cup\cap$ and the last visit. If there’s a way to solve the puzzle starting from $\cap\cup\cap$, we can implement it the first time we’re in state $\cap\cup\cap$.

This reasoning works in general, not just for vertex $\cap\cup\cap$, and not just for this puzzle. So a sequence of *distinct* vertices forming a walk can be an especially interesting sequence: in the context of a puzzle like the three cups puzzle, it represents a solution without obvious inefficiencies. This motivates the definition of a path.

Definition 3.2. A **path**² is a walk in which no vertex appears more than once. (An $x - y$ walk which is a path is called an $x - y$ path.)

¹To make the puzzle appear possible to solve, you can do the following: set up the puzzle in the alternating $\cup\cap\cup$ state, and make a few moves back and forth before arriving at the desired $\cap\cap\cap$. 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 $\cap\cup\cap$ state. After this, the puzzle will be impossible. You can further disguise the nature of the puzzle by using 5 cups instead of 3.

²I should mention that, although the terminology is fairly standard by now (the existence of Wikipedia, which has picked a side, helps) there are many sources only a couple of decades old which use a different convention. They say “path” when they mean “walk”, and then when they don’t want vertices repeated, say “simple path” when they mean “path”. I will not do this.

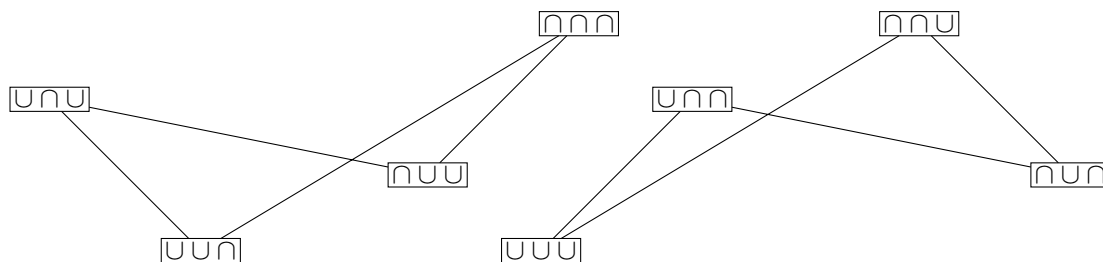


Figure 3.2: The two connected components of the three cup puzzle graph

In Chapter 2, we defined the path graph P_n : the graph with n vertices $\{1, 2, \dots, n\}$ and $n - 1$ edges $\{1, 2\}, \{2, 3\}, \dots, \{n - 1, n\}$. This is closely related to the concept of a path: whenever we have an n -vertex path in a graph G , it gives us a copy of the path graph P_n inside G . The vertices of the copy of P_n are the vertices along the path, and its edges are the edges that we know must exist between consecutive vertices. Going the other direction is also possible, more or less: every copy of the path graph P_n inside G corresponds to an n -vertex path. However, subgraphs isomorphic to P_n don't have a preferred direction of travel: an $x - y$ path and the $y - x$ path obtained by reversing it correspond to the same copy of a path graph.

Despite the slight inconsistency of whether we care about direction, it is often useful to switch between two ways of thinking about a path: as a specific kind of walk (the way we defined it) and as a specific kind of subgraph (a copy of P_n for some n). For example, if we define P to be the path $(x_0, x_1, x_2, \dots, x_n)$, it can later be convenient to refer to the edges $\{x_0x_1, x_1x_2, \dots, x_{n-1}x_n\}$ as $E(P)$, the “edges of P ”.

It does not make sense to think of arbitrary walks in G as subgraphs of G , however: we cannot recover the structure of the walk simply from knowing the vertices and edges used by the walk, because they can potentially be used multiple times and in many different orders.

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: for which x and y does an $x - y$ walk exist in the graph?

In the simplest scenario, the answer is always “yes”:

Definition 3.3. We say that a graph G is **connected** if an $x - y$ walk exists for all $x, y \in V(G)$.

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 $\{UUU, UUU, UUU, UUU\}$ and $\{UUU, UUU, UUU, UUU\}$.

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* confirm that it has no solution by checking each of the states UUU , UUU , UUU , and UUU 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 minimal effort—if we notice that the two halves of the graph are characterized by whether the number of \cup 's (cups which are right side up) is even or odd. A move either increases the number of \cup 's by two, leaves it the same, or decreases it by two; therefore it's impossible to go from an odd- \cup state like $\cup\cup\cup$ to an even- \cup state like $\cap\cap\cap$.

Though a split into *two* pieces with no edges between them is sufficient to show that a graph is not connected, 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 partition of the vertex set into connected components.

Definition 3.4. *If G is a graph, a **connected component of G** is a set $C \subseteq V(G)$ such that*

- *Whenever $x, y \in C$, there is a $x - y$ walk in G ;*
- *If $x \in C$ but $y \notin C$, there is no $x - y$ walk in G .*

We can also think of a connected component C as a subgraph: the induced subgraph $G[C]$. The first part of the definition above tells us that the subgraph $G[C]$ is connected; however, it follows from the second part of the definition that there are no edges leaving C , so there is no bigger connected subgraph containing C .

In the case of the three cup puzzle, the split into the left half and the right half of Figure 3.2 is in fact the split into connected components. The two halves, each isomorphic to a cycle graph C_4 , are connected subgraphs, so they are the two components of the three cup puzzle graph.

I've *claimed* that any graph can be broken up into connected components, but I've given you no proof of this. It is too easy to believe that such a thing 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 the set $V(G)$ 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

Let G be a graph. We define a relation \rightsquigarrow on pairs of vertices $x, y \in V(G)$: $x \rightsquigarrow y$ if there is an $x - y$ walk in G . I am not aware of an official name for this relation, but we might call it “reachability”: $x \rightsquigarrow y$ means that from x , we can reach y by following edges in G .

Question: In the three cup puzzle, for which vertices x does $x \rightsquigarrow \cup \cup \cap$ hold?

Answer: For the vertex $\cup \cup \cap$ itself, for its neighbors $\cup \cap \cup$ and $\cap \cap \cap$, and finally for the vertex $\cap \cup \cup$ that can reach $\cup \cup \cap$ in two steps.

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

Lemma 3.1. *The relation \rightsquigarrow 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). A relation \sim on a set S is an **equivalence relation** if it has three properties:

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 split up (or **partition**) S into equivalence classes: disjoint sets S_1, \dots, S_k such that $S = S_1 \cup \dots \cup S_k$ and we have $x \sim y$ exactly when x and y are in the same class. In the case of our relation \rightsquigarrow , the equivalence classes will give us the connected components of a graph.

Proof of Lemma 3.1. Let’s check all three properties of an equivalence relation.

We check that \rightsquigarrow 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 \rightsquigarrow 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, \dots, u_ℓ is an $x - y$ walk (with $u_0 = x$ and $u_\ell = y$). Then reverse it: $u_\ell, u_{\ell-1}, u_{\ell-2}, \dots, u_0$ 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 \rightsquigarrow is transitive. Suppose x, y, z are vertices in G such that there is an $x - y$ walk u_0, u_1, \dots, u_ℓ and a $y - z$ walk v_0, v_1, \dots, v_m . We know that $x = u_0$, $y = u_\ell = v_0$, and $z = v_m$. Then there is also an $x - z$ walk: the sequence

$$u_0, u_1, \dots, u_\ell, v_1, v_2, \dots, v_m.$$

This definitely starts at $x = u_0$ and ends at $y = v_m$. 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 $u_\ell = y = v_0$, and v_0v_1 are adjacent because they are consecutive in our $y - z$ walk.

Having checked all three properties, we know that \rightsquigarrow is an equivalence relation. \square

Using Lemma 3.1 and the properties of equivalence relations, we know that we can partition $V(G)$ into equivalence classes of \rightsquigarrow . These equivalence classes are sets C_1, C_2, \dots, C_k satisfying three properties:

1. They are pairwise disjoint: if $i \neq j$, then $C_i \cap C_j = \emptyset$.
2. Together, they include all the vertices: $C_1 \cup C_2 \cup \dots \cup C_k = V(G)$.
3. For $x, y \in V(G)$, we have $x \rightsquigarrow y$ if and only if there is a single C_i such that $x \in C_i$ and $y \in C_i$.

Property 3 of an equivalence class is exactly the property we need to know that C_i is a connected component of G . We conclude:

Theorem 3.2. *The vertices of any graph can be partitioned into connected components.*

When the graph is connected, and there is a walk from any vertex to any other vertex, then $V(G)$ itself is a single connected component.

This is not just interesting for answering questions about walks! Because there are no edges between different connected components, they basically do not interact with each other as far as graph properties are concerned. A lot of the time, if we're asking a question about graphs, we can work with each connected component separately.

Looking back at the examples in the first chapter, 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 the adjacency graph of 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, we get separate coloring problems for each island.

Or suppose we're looking for a largest **independent set**: a set of vertices with no edges between them. (This came up in the tiling problem from Chapter 1.) We can just find a largest independent set inside each connected component, then take their union.

Question: Suppose we want to find a largest **clique** in a graph: a set of vertices in which every pair is adjacent. How can we take advantage of knowing the connected components of the graph?

Answer: A clique cannot contain two vertices in different connected components. So we can first solve the problem for each connected component, finding a largest clique in each one. Then, take whichever clique is largest among the ones we found.

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. For this, the following definition is a starting point:

Definition 3.5. *A walk is **closed** if it starts and ends at the same vertex.*

Closed walks with no further conditions imposed on them are not very notable, because they're very easy to come by. For example, we can take an arbitrary walk and follow it up with the reverse of that walk, tracing all our steps back to the start. This is a closed walk that is not very satisfying to find, because it reveals nothing about the structure of our graph.

(Why then, do we make the definition at all? One reason is that it is a building block for later definitions, but another reason is that the matrix techniques in Appendix C make walks and closed walks especially easy to count. So we had better have a name for what it is we're counting!)

We'd like to make a definition of a closed walk that is interesting to come by: that heads out and comes back by a different route. The most restrictive and most useful such definition is that of a cycle:

Definition 3.6. *A closed walk $x_0, x_1, x_2, \dots, x_k$ is a **cycle** if $k \geq 3$, and all of the vertices x_0 through x_{k-1} are distinct (but $x_k = x_0$, because the walk is closed).*

Associated to the cycle $x_0, x_1, x_2, \dots, x_k$ with $x_k = x_0$ is a subgraph isomorphic to C_k : the subgraph which has vertex set $\{x_0, x_1, x_2, \dots, x_{k-1}\}$ and edge set $\{x_0x_1, x_1x_2, \dots, x_{k-1}x_k\}$. It is in part to make this connection work that we require $k \geq 3$ in the definition of a cycle: the closed walks with $k = 1$ or $k = 2$ would not really be “cycle-like” objects. Just as with paths, we will often flip back and forth between two ways of thinking about a cycle: as a sequence of vertices, and as a subgraph.

(This seems very handwavy and imprecise, so it's important to be very clear when we are doing this. On the other hand, it is also very rewarding to be able to look at the same object from two different points of view.)

<p>Question: If we think of a path as a subgraph instead of a walk, we lose track of the direction: which endpoint is the start and which is the end. What do we lose track of when we think of a cycle as a subgraph?</p>
<p>Answer: Two things. First, in the closed walk definition of a cycle, there is still a direction: we go from x_0 to x_1, not vice versa. This is lost when we switch to the subgraph. Second, the vertex x_0 which is both the start and the end plays a special role in the closed walk definition; in the subgraph, all vertices are treated equally.</p>

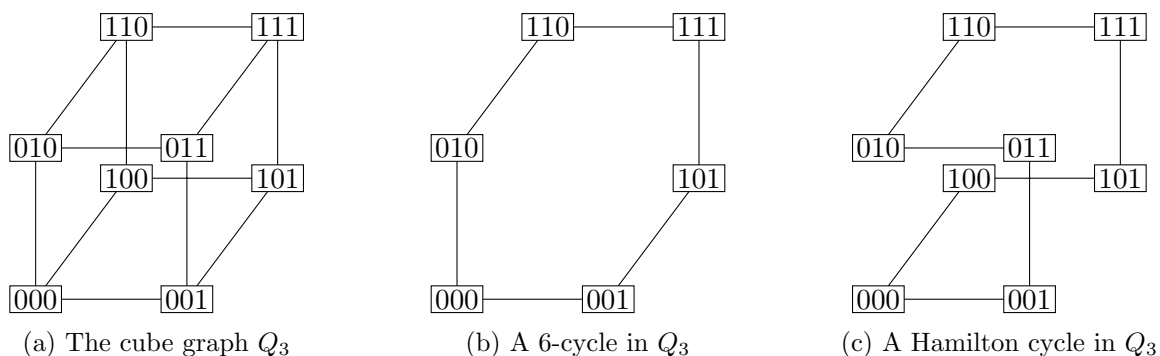


Figure 3.3: The cube graph and some cycles in it

In Figure 3.1, the closed walk

$$\cup \cup \cup, \cup \cap \cap, \cap \cup \cap, \cap \cap \cup, \cup \cup \cup$$

is 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 \cup and \cap , 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, later on, 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.

In the cube graph, we also have 4-vertex cycles such as 000, 001, 011, 010, 000. We also have 6-vertex cycles such as the one shown in Figure 3.3b, and even cycles through all 8 vertices such as the one shown in Figure 3.3c. (A cycle through all the vertices of a graph is called a **spanning cycle** or **Hamilton cycle**: viewed as a subgraph, it really is a spanning subgraph, so this is in line with our past definitions.)

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 3.7. A walk $x_0, x_1, x_2, \dots, x_k$ has *length* k .

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 corresponds to a cycle of length n , but the path graph P_n corresponds to a path of 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 notion of length is a useful one to have before we prove the following theorem, which tells us when *paths* exist in a graph:

Theorem 3.3. *Let x, y be two vertices of a graph G . If there is an $x - y$ walk in G , then there is a $x - y$ path in G , as well. Moreover, a shortest $x - y$ walk (one with the smallest possible length) is always a path.*

Question: Why do mathematicians say “a shortest walk” when it would sound more natural to say “the shortest walk”?

Answer: Referring to “the shortest $x - y$ walk” could be misinterpreted to mean that there is only one $x - y$ walk which is the shortest. This is often false.

Proof of Theorem 3.3. Because we are discussing an object that minimizes something (length, in this case), it’s natural to use the extremal principle. Assuming $x - y$ walks exist in G , let

$$u_0, u_1, u_2, \dots, u_k$$

(with $u_0 = x$ and $u_k = y$) be an $x - y$ walk of minimum length. We will show that this walk is actually a path, which proves both halves of the theorem.

Suppose for the sake of contradiction that the walk is not a path; in that case, 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

$$u_0, u_1, \dots, u_{i-1}, u_i, u_{j+1}, \dots, u_k$$

in which we skip vertices $u_{i+1}, u_{i+2}, \dots, u_j$. 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 $k - (j - i)$: strictly less than k . This contradicts our assumption that we took an $x - y$ walk of minimum length. So a shortest $x - y$ walk cannot have repeated vertices: it must be a path. \square

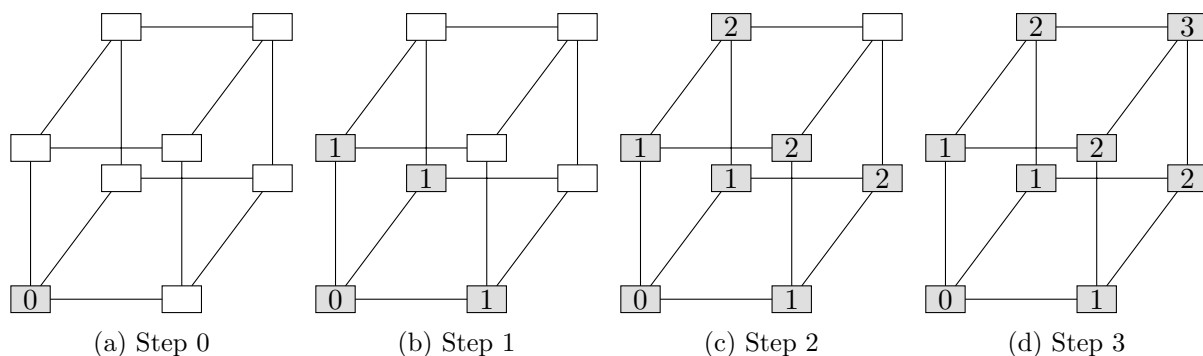


Figure 3.4: Finding distances in the cube graph

3.6 Distances

Having proven Theorem 3.3, we can now define distances in a graph.

Definition 3.8. If x and y are vertices in a graph G , the **distance between x and y** (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 3.4d, 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 path starting and immediately ending at 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 $d(x, x) = 0$. We will consider that starting point to be “Step 0” of the algorithm; it is shown for $G = Q_3$ in Figure 3.4a.

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 3.4c.

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, \dots, x_j with $x_0 = x$ and $x_j = z$. The walk x_0, x_1, \dots, 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 3.4c and the result of Step 3 is shown in Figure 3.4d. 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 $d(x, z)$ a step after computing $d(x, y)$ if not earlier.

Therefore 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.

3.7 Practice problems

1. Let G be the graph whose vertices are the numbers $1, \dots, 15$, 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 .

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 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 4.5 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.

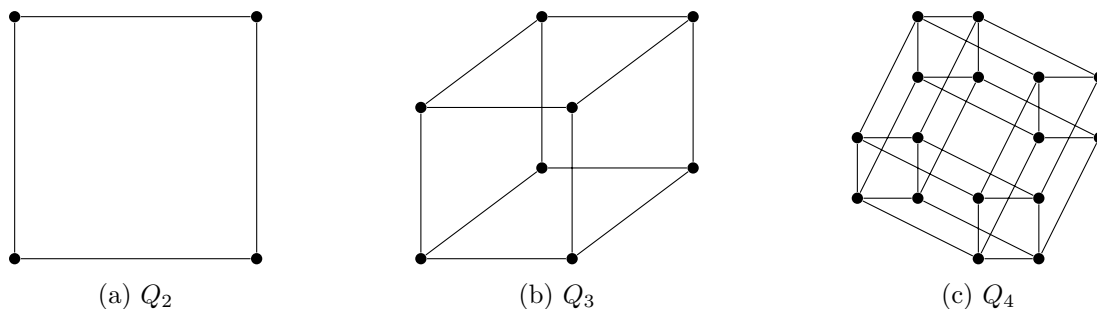


Figure 4.1: Three hypercube graphs of three different dimensions

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 $\{1, 2, \dots, n\}$, where an n -bit string $b_1b_2 \dots b_n$ 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 $b_1b_2 \dots b_n \in Q_n$: 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 x in a graph G is the number of edges incident on x (having x as an endpoint). We write $\deg_G(x)$ for the degree of vertex x in G , or just $\deg(x)$ if the graph is clear from context.

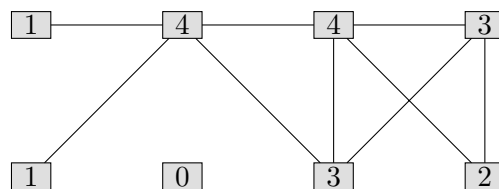


Figure 4.2: An 8-vertex graph labeled with the degrees of its vertices

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. A vertex with degree 0 is called an **isolated vertex**, and a vertex with degree 1 is sometimes called a **leaf**.

Isolated vertices are interesting because they are the smallest possible connected component in a graph. Leaves are less obviously notable; we will (appropriately) find out what’s interesting about them when we study tree graphs.

Definition 4.4. The **maximum degree** of a graph G , denoted $\Delta(G)$, is the largest degree of any vertex; the **minimum degree** of a graph G , denoted $\delta(G)$, 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, which we’ll prove in Chapter 14.)

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 inductive 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 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 inductive 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 Lemma 4.1, 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 $b_1b_2 \dots b_n$ has n places in which a bit can 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.) □

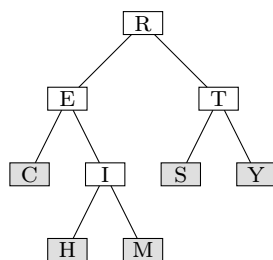


Figure 4.3: An full binary tree with 8 vertices; the leaves are shaded.

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! \square

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, with 0 children. 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. \square

We can generalize the answer to Problem 4.1. Let's say that a vertex v is **even** if $\deg(v)$ is an even number, and **odd** if $\deg(v)$ is an odd number. Then:

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

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 vertices is even, the parity is flipped an even number of times, and ends at an even number; if the number of odd vertices 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, the number of odd vertices must be even. \square

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 $x - y$ path.

This is definitely true if xy is an edge of G : in this case, the sequence x, y is an $x - y$ path. 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 $x - y$ path, completing our proof. \square

(Actually, we've shown more: we've shown that for any two vertices $x, y \in V(G)$, the distance $d(x, y)$ is at most 2. We say that G has **diameter** at most 2. This term is defined by analogy to a circle, in which the diameter is the longest distance between any two points on the circle. For a graph, the diameter is the longest distance between any two vertices of the graph.)

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 $\delta(G) \geq \underline{\hspace{1cm}}$ 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 an **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 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 4.6 in the next section, will both be very useful to us multiple times in future chapters. 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.*

The intuition for this theorem is as follows. Just start at any vertex of G and walk around, taking care not to leave a vertex the way you entered it. Eventually you will run into a vertex you’ve seen before. The first time that happens, your trajectory from that vertex and back forms a cycle.

Whenever we have an intuition of the form “keep doing this thing until it does what we want”, this suggests a proof with the extremal principle. Just take (by the extremal principle) the situation in “this thing” has been done for as long as it possibly can without “doing what we want”. Then it is forced to “do what we want” in the very next step.

In this particular case, if we walk around for as long as we possibly can without revisiting a vertex, what we’re getting is a very long path. This suggests the following proof:

Proof. Let $x_0, x_1, x_2, \dots, x_k$ be a longest path in G .

We know $\delta(G) \geq 2$ and therefore in particular $\deg(x_0) \geq 2$. Is it possible that x_0 has a neighbor y which is *not* one of x_1, \dots, x_k ? It's not! In that case, y, x_0, x_1, \dots, x_k would be a longer path.

So x_0 has at least two neighbors, which are all in the set $\{x_1, x_2, \dots, x_k\}$. One of x_0 's neighbors is x_1 : the next vertex on the path. This doesn't help us. But there must be another vertex x_i with $i \geq 2$ which is adjacent to x_0 .

Then $(x_0, x_1, \dots, x_i, x_0)$ is 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 saying, "Let $x_0, x_1, x_2, \dots, x_k$ be 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 $x_0, x_1, x_2, \dots, x_k$ be a longest walk in G ," because there are often arbitrarily long walks.

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 [3] to solve a problem relating average degree to the existence of certain subgraphs, though special cases of it were used even earlier.

Lemma 4.5. *Let G be a graph with average degree at least d , where d is a positive integer. Then G contains a subgraph H with $\delta(H) > \frac{d}{2}$.*

This *could* be done using another application of the extremal principle: if you pick a subgraph H with the highest average degree, then it turns out to work. Rather than do this, we will use induction, so that we get to practice induction. (Take note of how we structure this proof to avoid the "induction trap"!)

Proof. We induct on n , the number of vertices in G .

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, \dots, x_n , then the average of the degrees is

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

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 inductive 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 4.5, we can convert minimum-degree results to average-degree results. For example, we can use Lemma 4.5 to turn Theorem 4.4 into a result about average degree—or, equivalently, about the number of edges. This will be very valuable to us.

Corollary 4.6. *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 4.5, 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 4.6 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 are:

- The tetrahedron, which has 4 vertices and 4 triangular faces, with 3 faces meeting at every corner.
- The cube (or hexahedron), 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. (*In Chapter 23, we will explore further constraints on the Platonic solids.*)

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 4.6 on the existence of cycles by combining Theorem 4.4 and Lemma 4.5.

In a similar way, we can combine Proposition 4.3 on connectedness and Lemma 4.5 to obtain a result about the minimum average degree (or, equivalently, the minimum number of edges) to 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 4.5, 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. 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 [2] for a solution to the problem! That gives you an idea of how hard that would be.)

Bibliography

- [1] 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](https://doi.org/10.3390/robotics11010016).
- [2] John H. Conway. *Five \$1,000 Problems (Update 2017)*. 2017. URL: <https://oeis.org/A248380/a248380.pdf> (visited on 09/23/2025).
- [3] 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](https://doi.org/10.4064/cm-13-2-251-254).
- [4] KarzA. *Vectormap of Switzerland with all canton, names and flags as separated layers*. 2008. URL: https://en.wikipedia.org/wiki/File:Kantone_der_Schweiz.svg (visited on 09/23/2025).
- [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.