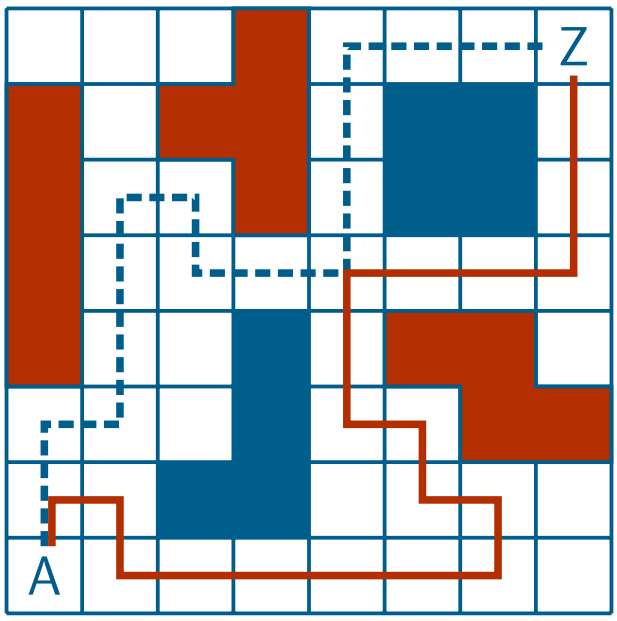
Mikhail Lavrov

Start Doing Graph Theory

Part VII: Connectivity



available online at https://vertex.degree/

Contents

About this document

Where it's from

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

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

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

What's inside

In this part of the book, I cover topics related to the connectivity of graphs: how resilient they are to being disconnected by removing vertices or edges. Chapter 25 introduces cut vertices: one of the simplest special cases. Chapter 26 covers vertex and edge connectivity in general, motivating Menger's theorem, which is stated and proved in Chapter 27. Finally, Chapter 28 is more algorithmic: it covers the maximum flow problem, which (in addition to its applications outside graph theory) can be used to compute the connectivity of graphs.

The cover

The cover of this PDF shows two paths from A to Z in the path-counting problem I use to motivate cut vertices in Chapter 25. How many other paths are there?

The license

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

25 Cut vertices

The purpose of this chapter

With this chapter, we embark on the topic of connectivity: exploring how many vertices or edges must be deleted from a graph to make it disconnected, or disconnected in some specific way.

This is the final part of the textbook, so I will spend more time pointing out the connections between the new topics covered in each chapter and other parts of graph theory. (For this chapter, none of the previous topics are critical for understanding; they only serve as examples. Later on, this will change.)

As a graph theorist, I naturally have a graph-theoretic explanation of why pointing out these connections is important. I think of mathematical concepts as forming a very large graph of ideas in my head, with edges representing these connections. This is an undirected graph; in this textbook, I go through some of its vertices in a specific order, but if two ideas are related, either one helps understand the other.

Sometimes you forget things, and the graph of ideas loses a vertex. This is natural. But you want your graph to be resilient to forgetting things. If you lose a vertex with many neighbors, then you can easily recover: you can remember the adjacent ideas and how they related to what you forgot. Don't let your graph of ideas have cut vertices!

25.1 Counting paths

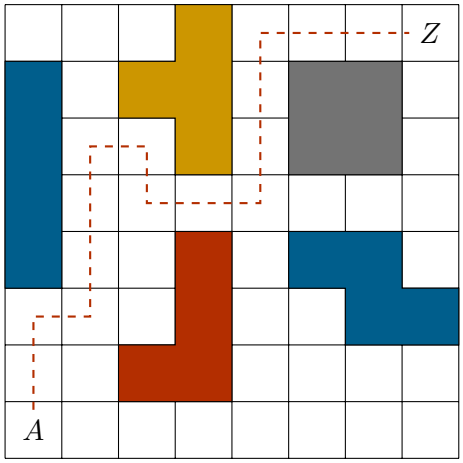
Let me begin with a puzzle.

Problem 25.1. *In Figure 25.1a, several Tetris pieces have been placed on an 8×8 grid. How many ways are there to get from point A (the bottom left corner) to point Z (the top right corner) by a path through the square in the grid that does not pass through any Tetris pieces and does not visit a square more than once?*

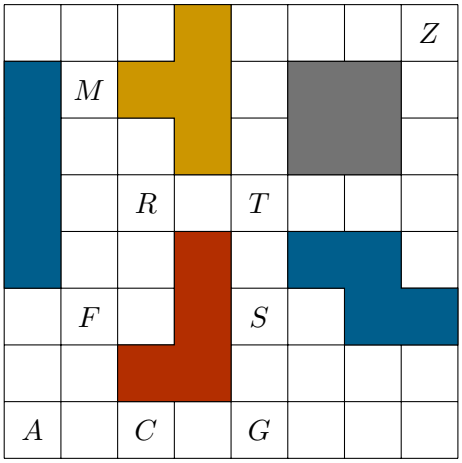
This is a book, so I can't ask you to think about the puzzle on your own for a bit before I reveal the answer. However, I really do think it's a good idea for you to think about it: you'll internalize the ideas in this chapter better if you come up with some of them on your own first. You should especially try to think of a way to solve the problem systematically and with as little brute force as possible: there's many paths, so you don't want to just list them all.

When you've done all the thinking you want to do, keep reading.

No rush—take your time.



(a) The puzzle, with one sample path



(b) A few useful squares

Figure 25.1: Count the paths from A to Z!

Alright! So: the first insight that cuts down on the work required is that the square marked T in Figure 25.1b is special. All paths from A to Z have to go through T (and they are only allowed to go through T once). So every $A-Z$ path is the union of an $A-T$ path and a $T-Z$ path; conversely, the union of every $A-T$ path and every $T-Z$ path is an an $A-Z$ path. This means that to get our final count, we can multiply the number of paths from A to T by the number of paths from T to Z.

Question:	Which graph are all these paths in?
Answer:	The graph obtained from the 8×8 grid graph by deleting every vertex covered by a Tetris piece. Squares like A, T, and Z are vertices of this graph.

Let's make up some notation, just for this problem: let $p(x, y)$ be the number of paths from x to y. In this notation, we are looking for $p(A, Z)$, but we've just discovered that $p(A, Z) = p(A, T) \cdot p(T, Z)$.

Question:	What is $p(T, Z)$?
Answer:	It is just 2: from T, we can go up and then right, or right and then up, but the grey square Tetris piece stops us from doing anything else.

Next, we can simplify $p(A, T)$ with just a bit of casework. We can arrive at T either from the left (through the square marked R) or from below (through the square marked S). So we can count both kinds of paths separately.

Question:	Is it true that $p(A, T) = p(A, R) + p(A, S)$?
Answer:	No: as we've defined it, $p(A, R)$ counts all paths from A to R, including ones that go through T first—and we can't follow those up by going back to T.

Instead, let's define a second function $q(x, y)$, to be the number of paths from x to y that don't go through T. Now it's okay to say that $p(A, T) = q(A, R) + q(A, S)$. Let's compute $q(A, R)$ first, and come back to look at $q(A, S)$ second.

Without going through T, every path from A to R must pass through F first: like our initial identity, this lets us factor $q(A, R)$ as $q(A, F) \cdot q(F, R)$. What's more, the paths cannot go through squares C or M: if they do, there's no way to return! So $q(A, F)$ can be computed just by looking at a 3×2 rectangle in the bottom left corner, and $q(F, R)$ can be computed just by looking at a 4×2 rectangle. I have no more tricks to suggest here, but there's not many paths, so we can just count them: $q(A, F) = 4$ and $q(F, R) = 6$.

When we move on to $q(A, S)$, the role of certain squares is reversed. The path cannot go through F, because that cuts off our path of retreat. It must go through C, and then it must go through G. (There is only one way to get from C to G at this point.) So we can factor $q(A, S)$ as $q(A, C) \cdot q(G, S)$. Once again, these are small problems we can solve in just one corner of the grid by listing out all the paths; you should get $q(A, C) = 3$ and $q(G, S) = 8$.

We are now ready to combine the answers we got:

$$\begin{aligned} p(A, Z) &= p(A, T) \cdot p(T, Z) \\ &= \Big(q(A, R) + q(A, S)\Big) \cdot p(T, Z) \\ &= \Big(q(A, F) \cdot q(F, R) + q(A, C) \cdot q(G, S)\Big) \cdot p(T, Z) \\ &= \Big(4 \cdot 6 + 3 \cdot 8\Big) \cdot 2 = 96. \end{aligned}$$

25.2 Cut vertices

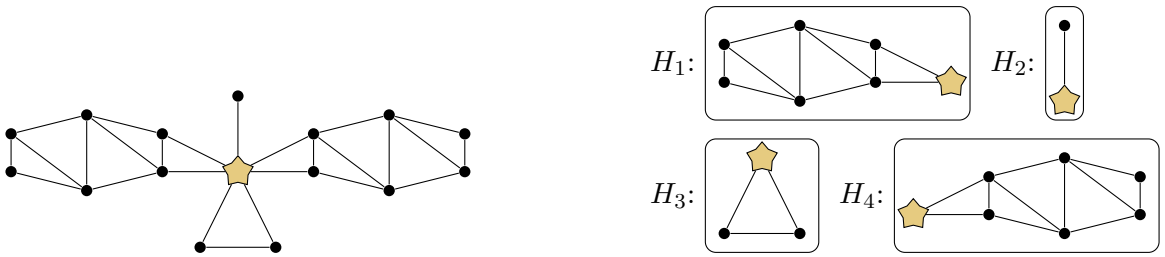
The graph-theoretic concept at work in our solution to Problem 25.1 is the notion of a cut vertex.

A **cut vertex** x of a graph G is a vertex such that $G - x$ has more connected components than G. Most commonly, G is a connected graph, in which case x is a cut vertex if and only if $G - x$ is not connected.

For example, in the graph representing Problem 25.1, vertex T is a cut vertex: deleting it separates A from Z. (Several other vertices marked in Figure 25.1b, such as F and G, are cut vertices of subgraphs of this graph.)

The definition of a cut vertex might resemble the definition of a bridge given in Chapter 9: we defined a bridge to be an edge that disconnects a graph (or increases the number of connected components) when we remove it. For this reason, bridges are sometimes called cut edges.

Question:	We know that in a tree, every edge is a bridge. Which vertices in a tree are cut vertices?
Answer:	The cut vertices are exactly the vertices which are not leaves. When a leaf is deleted, a smaller tree is left; however, deleting a vertex of degree $k > 1$ leaves k connected components, one for every neighbor of the deleted vertex.



(a) A cut vertex in G (b) H_1, H_2, H_3, H_4

Figure 25.2: Using a cut vertex to break G down into $H_1, H_2, \ldots, H_k$

What are cut vertices good for? It's hard to give a complete answer to a question like that in graph theory, because an easy-to-define concept can pop up in seemingly unrelated questions. On their own, cut vertices are good in applications because they tell us about the reliability of computer networks [9] or supply networks [3], just to name two examples. Usually, identifying the cut vertices does not tell the complete story; in this part of the textbook, you will learn some more complicated measures of reliability, as well.

It can also be helpful to know when a graph has a cut vertex no matter which problem we're trying to solve, because it can help us break down the problem into smaller and simpler problems: we saw that in action in Problem 25.1. Let's be a bit more specific about this, though. Given a connected graph G, suppose that x is a cut vertex of G. Where do we find the subproblems?

Well, we could start by identifying $G_1, G_2, \ldots, G_k$, the connected components of $G - x$. But that's not usually quite what we want, because none of these connected components "remember" how they fit together. Instead, we want subgraphs $H_1, H_2, \ldots, H_k$ that all come together at x. Formally, let H_i be the subgraph of G induced by $V(G_i) \cup \{x\}$; this subgraph includes only one connected component of $G - x$, but also keeps track of how that component is attached to x. An example is shown in Figure 25.2.

Let's look at some examples to see how this helps us solve a few of the problems that have been covered so far in this book.

Question: How does determining whether $H_1, H_2, \ldots, H_k$ are planar help us decide if G is planar?

Answer: If all k subgraphs are planar, their plane embeddings can be combined by placing them around x like the petals of a rose, so G is also planar; Figure 25.2 is an example of this. If one of the subgraphs is not planar, then G as a whole isn't planar, either.

Question: How does coloring the graphs $H_1, H_2, \ldots, H_k$ help us find a coloring of G?

Answer: Though the colorings are not necessarily compatible "out of the box", we can permute the colors used on each subgraph so that all of them give x the same color. Then, by combining the colorings, we get a coloring of all of G.

In particular, $\chi(G) = \max\{\chi(H_1), \chi(H_2), \ldots, \chi(H_k)\}$.

Question:	How does finding the largest clique in $H_1, H_2, \ldots, H_k$ help us find the largest clique in G?
Answer:	A clique in G must be entirely contained in some H_i: if we take two vertices other than x from different subgraphs, then they won't be adjacent. In particular, $\omega(G) = \max\{\omega(H_1), \omega(H_2), \ldots, \omega(H_k)\}$.

Question:	How does finding a spanning tree of $H_1, H_2, \ldots, H_k$ help us find a spanning tree of G?
Answer:	We can just take the union of the spanning trees of all the subgraphs. This is always a spanning tree of G, and every spanning tree of G is obtained in this way, so this can be used to count the spanning trees of G, or to find a minimum-cost spanning tree.

There are many more examples, but let's move on.

25.3 2-connected graphs

What happens if we split up a graph at its cut vertices (as in the previous section) as many times as possible? Eventually, we arrive at a bunch of smaller graphs with no more cut vertices.

We'll probably have to use some other tools to solve whatever problem we were trying to solve in those small graphs. However, maybe the property of having no cut vertices will make them special somehow? We should study graphs with no cut vertices to see if we can learn anything useful about them.

(From the point of view of reliability of a network, graphs with no cut vertices are also special: they are the graphs that stay connected even if something happens to one of the vertices.)

I'm going to do something a bit strange with the definition here, and say that a 2-*connected* graph is a connected graph with at least 3 vertices and no cut vertices.[1] Why is that clause "at least 3 vertices" there?

It's possible to have a 2-vertex graph with no cut vertices: the graph K_2. (We even saw K_2 among the smaller graphs we got in Figure 25.2.) However, in many ways, K_2 is unusual for graphs with no cut vertices. The reason is that we can rephrase the definition as saying "For any three different vertices x, y, and z, if x is deleted, there is still a path from y to z." A graph that doesn't have three different vertices satisfies this definition in a trivial way.

Many of the theorems we prove about 2-connected graphs will be false for K_2, which is why we exclude it from the definition. This starts with the following theorem, which would be false if K_2 were considered to be a 2-connected graph:

[1] I am also making this an italicized, "minor" definition, because in Chapter 26, we will define what it means for a graph to be *k-connected*, and will not need to refer to this special case again.

Theorem 25.1. *A graph G is 2-connected if and only if any two vertices of G lie on a common cycle. (That is, for all $x, y \in V(G)$ there exists a cycle in G through both x and y.)*

Theorem 25.1 is an if-and-only-if result, so it is saying two things:

1. If any two vertices of G lie on a common cycle, then G is 2-connected.
2. If G is 2-connected, then any two vertices of G lie on a common cycle.

Of these, statement 1 is the easier of the two directions to show, so we will go ahead and prove it right away; we will return and prove the second direction of the theorem later in this chapter.

Proof of half of Theorem 25.1. Suppose that any two vertices of a graph G lie on a common cycle. (In particular, G must contain at least one cycle, which means that there must be at least 3 vertices.)

Let x be any vertex of G. To verify that $G - x$ is connected, let y and z be two vertices of $G - x$. How do we show that in $G - x$, there is a $y - z$ path?

Well, in G, there is a cycle containing both y and z. We can represent this cycle by a closed walk

$$(x_0, x_1, \ldots, x_{l-1}, x_0)$$

where $x_0 = y$ and $x_i = z$ for some i. This walk can be split in two:

$$(x_0, x_1, \ldots, x_{i-1}, x_i) \quad \text{and} \quad (x_0, x_{l-1}, \ldots, x_{i+1}, x_i).$$

Both of these walks represent $y - z$ paths, and they have no vertices other than y and z in common. Therefore x (the vertex we delete) can only appear as a vertex on one of the paths, which means that the other $y - z$ path survives to $G - x$. □

Through Theorem 25.1, 2-connected graphs have some ties to topics covered earlier in this book. Here's a quick example:

Proposition 25.2. *If G is a Hamiltonian graph, then it is 2-connected.*

Proof. If G has a Hamilton cycle, then any two vertices of G lie on that cycle, so G is 2-connected by Theorem 25.1. □

Question:	In fact, Proposition 25.2 is a special case of an earlier result about Hamiltonian graphs. How?
Answer:	From Corollary 17.3, we know that all Hamiltonian graphs are tough: if $k \geq 1$ vertices are deleted, at most k connected components are left. In the special case $k = 1$, this tells us that there cannot be any cut vertices.

Much of the early work on 2-connected graphs and connectivity in general was done by Hassler Whitney, who proved Theorem 25.1, Theorem 25.4, Proposition 25.5, and Theorem 26.3 in 1932 [12, 13].

25.4 Ear decompositions

Many times in this textbook, we've discussed the following kind of question: if you've solved a graph-theoretic problem, how do you give a short but convincing demonstration that your solution is correct? Let's recap a few instances of this:

- If two graphs are isomorphic, then we can write down an isomorphism, and it will be tedious but straightforward to check that it is an isomorphism.

  If two graphs are not isomorphic, a quick way to demonstrate this is by finding a graph invariant that the two graphs disagree in. However, finding such an invariant might be hard.

- If we want to know the matching number $\alpha'(G)$ of a bipartite graph, and we've found a large matching M, that's only a proof that $\alpha'(G) \geq |E(M)|$. What if there's a larger matching? But by Kőnig's theorem (Theorem 14.2), we can always find a vertex cover U with $|E(M)| = |U|$, and use it as a proof that M is as large as possible.

- If a graph is Hamiltonian, we might still have to work very hard to find a Hamilton cycle; but once we've found it, it is very easy to check.

  What do we do if a graph G is not Hamiltonian? Sometimes, if we're lucky, the graph will not be *tough* (as defined in Chapter 17), either. In that case, after yet more hard work, we can find a set $S \subseteq V(G)$ such that $G - S$ has more than $|S|$ connected components, proving that G is not tough and not Hamiltonian. But there are graphs for which this will not work; we might not always have a short proof.

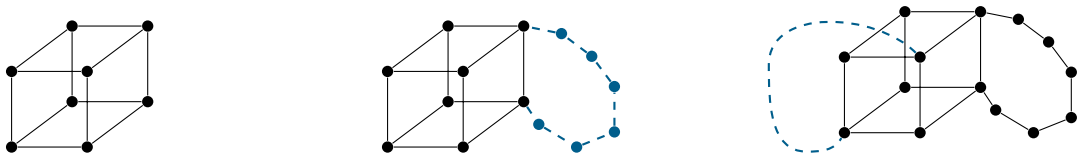
- If a graph is planar, then we can demonstrate this by drawing a plane embedding. What if the graph is not planar? Kuratowski's theorem (Theorem 22.7) tells us that we can demonstrate this by finding a subdivision of $K_{3,3}$ or K_5 inside the graph. Finding the subdivision might take some work, but checking it takes much less work; it's great for busy teachers grading graph theory homework.

These ideas are not just important when applied to individual graphs. They also often come up when writing a proof! Often, proving that something does exist is a lot easier than proving that something does not exist. The ideas here tell us how we can turn the second kind of proof-writing back into the first kind.

Let's think about how the idea of short demonstrations applies to 2-connected graphs.

Question:	Suppose a connected graph G is not 2-connected. How can you demonstrate this?
Answer:	You could find a cut vertex x.

Question:	Okay, but how do you demonstrate that x is a cut vertex?
Answer:	You'd need to show that $G - x$ is not connected; Lemma 3.2 can be a useful tool for this.



(a) The cube graph (b) The cube, plus an ear (c) The cube, plus two ears

Figure 25.3: Adding ears to a cube graph

It's less straightforward to consider the other direction: if a graph is 2-connected, and you want to prove it, how do you do so? Working directly from the definition, you could check for every vertex x that $G - x$ is connected, perhaps by giving a spanning tree of $G - x$ (but verifying that a graph is a tree also takes some work). We could also try to find a collection of cycles so that every pair of vertices lies on a common cycle in our collection, and apply Theorem 25.1. However, we might need a lot of cycles to make this happen, and checking them will not be easy.

All this is a lengthy introduction to a new idea: ears, and ear decompositions.

Let an **ear** of a graph G be a path in G in which every vertex except the first and last has degree 2. From now on, we will also refer to the vertices of a path except the first and last as the **internal** vertices of the path, to make them easier to refer to.

When we **add an ear** to a graph G, we pick two different vertices x_0 and x_l already in $V(G)$; we add new vertices $x_1, x_2, \ldots, x_{l-1}$ and new edges $x_0x_1, x_1x_2, \ldots, x_{l-1}x_l$. This creates the ear represented by the walk $(x_0, x_1, x_2, \ldots, x_l)$.

It's easier to show what it means to add an ear by means of a picture, rather than with words. Figure 25.3 shows how, starting with a cube graph (Figure 25.3a), we add an ear to arrive at Figure 25.3b. This particular ear has many internal vertices, but that's not necessary. Adding an edge to a graph (as in Figure 25.3c) is a special case of adding an ear: it's adding an ear of length 1.

The following lemma is the reason why we introduce ears.

Lemma 25.3. *If G is a 2-connected graph and we add an ear to G, the resulting graph is also 2-connected.*

Proof. Let H be a graph obtained from G by adding an ear R: an $x - y$ path for two different vertices $x, y \in V(G)$ whose internal vertices only exist in H. We will prove that the new graph H still does not have a cut vertex.

To do this, we see what happens when we delete a vertex of H:

- Suppose we delete a vertex $z \in V(G)$ other than x or y. Because $G - z$ is connected, all vertices of $G - z$ are in the same connected component of $H - z$. Also, vertices of R all have a path to x and y (because R is connected), so they are also in that same connected component: $H - z$ is connected.

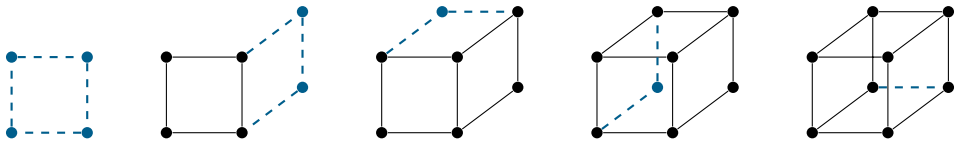


Figure 25.4: An ear decomposition of the cube graph

- If we delete x, essentially the same thing happens. The only change is that an internal vertex of the ear we added to G has only only one path to the connected component of $G - x$: its path to y in $R - x$. Swapping the roles of x and y, the same thing happens if we delete y.
- If we delete a vertex z which is an internal vertex of R, then G remains connected, and $R - z$ is divided into two path components: one containing x and one containing y. Every internal vertex of R other than z still has a path to either x or y, so it is in the same connected component as the vertices of G.

In all cases, H remains connected when we delete a vertex, so it is 2-connected. □

In particular, if we start with a cycle and repeatedly add ears, the result will be connected by Lemma 25.3 and by induction. We can use this to give a short proof that a graph G is 2-connected, if we can find a way to build G from a cycle inside G by adding ears. Figure 25.4 shows a way to do this for the cube graph Q_3.

We refer to a demonstration of 2-connectedness via Lemma 25.3 as an ear decomposition. Since we'd like to use ear decompositions in proofs, let's give a formal definition. An **ear decomposition** of a graph G is a sequence $R_1, R_2, \ldots, R_k$ of subgraphs of G with the following properties:

1. R_1 is a cycle.
2. R_i is an ear of $R_1 \cup R_2 \cup \cdots \cup R_i$ for all $i > 1$.
3. Each edge of G is in exactly one of the subgraphs R_i.

   As a consequence, $G = R_1 \cup R_2 \cup \cdots \cup R_k$.

(The last of these three properties is, as you might remember, exactly what we mean by a decomposition in general.)

Does such a proof always exist? Yes!

Theorem 25.4. *A graph G is 2-connected if and only if it has an ear decomposition.*

Proof. In both directions of this proof, we will have a sequence $R_1, R_2, \ldots$ of subgraphs of G, which we will either assume to be an ear decomposition or prove to be all or part of one. To simplify notation, let's write G_i for the union $R_1 \cup R_2 \cup \cdots \cup R_i$. In the case of an ear decomposition, R_i should be an ear of G_i, and we obtain G_{i+1} from G_i by adding the ear R_{i+1}.

If G has an ear decomposition, then it is 2-connected by repeated use of Lemma 25.3 in a short induction. Let the sequence $R_1, R_2, \ldots, R_k$ be the ear decomposition; then we prove by induction on i that G_i is 2-connected. The base case $i = 1$ holds because cycles are 2-connected:

deleting any vertex from R_1 leaves a path. Since G_{i+1} is obtained from G_i by adding an ear, Lemma 25.3 is exactly the induction step we need. Finally, $G_k = G$, so reaching $i = k$ proves that G is 2-connected.

To prove the other direction of the if-and-only-if statement, we have to reason in the opposite way from Lemma 25.3, building up the ear decomposition $R_1, R_2, \ldots, R_k$ step by step.

We can start by letting R_1 be any cycle in G. Why does a cycle exist? Well, we know G is connected and has at least three vertices. If G contained a leaf x, then the only neighbor of x would be a cut vertex, separating x from the rest of the graph. This is not allowed, so G must have minimum degree at least 2, giving it a cycle by Theorem 4.4.

Question:	Doesn't this also follow from Theorem 25.1?
Answer:	It does, but I want to avoid that argument. We are soon going to use ear decompositions to prove Theorem 25.1, and I want to make sure that proof is not circular.

Next, suppose we've gotten partway through the process, finding the subgraphs $R_1, R_2, \ldots, R_i$. These should satisfy the first two properties of an ear decomposition. Because we're not done yet, they don't have to satisfy the third property. However, we assume that $R_1, R_2, \ldots, R_i$ share no edges, and that $G_i = R_1 \cup R_2 \cup \cdots \cup R_i$ is a subgraph of G. We would like to make G_i bigger by adding an ear, which we'll call R_{i+1}; however, we want to add an ear that's still entirely contained in G.

The first question we ask is this: is $V(G_i) = V(G)$? If so, then we're nearly done, and continuing the ear decomposition is easy. Pick any edge $xy \in E(G)$ that is not in G_i, and make that edge (or, more precisely, the subgraph consisting of x, y, and edge xy) be the ear R_{i+1} we add next.

Question:	What will steps $i+2, i+3, \ldots$ look like, in this case?
Answer:	Since $V(G_{i+1}) = V(G)$ as well, we'll stay in this case until the end. We will add the remaining edges of G, one at a time, as ears of length 1.

So it's the early steps that we have to worry about, when $V(G_i)$ is still not all of $V(G)$. In this case, let xy be any edge where $x \in V(G_i)$ and $y \notin V(G_i)$; we will try to construct an ear that begins with the edge xy.

Question:	How do we know such an edge xy exists?
Answer:	If there is no such edge, then G would not have any edges between $V(G_i)$ and its complement $V(G) - V(G_i)$, so it would not be connected.

Since G is 2-connected, in particular $G - x$ is connected, so we can find a path in $G - x$ from y to any other vertex. Choose that other vertex to be a vertex $z \in V(G_i)$, not equal to x, of course. There is a $y - z$ path in $G - x$; represent it by a walk $(x_1, x_2, \ldots, x_l)$ with $y = x_1$ and $z = x_l$. By setting $x_0 = x$, we get an walk $(x_0, x_1, x_2, \ldots, x_l)$; this walk represents an $x - z$ path that starts and ends in G_i.

Question:	Is this path an ear we can add to G_i?
Answer:	Not necessarily.

Question:	What could be the matter with it?
Answer:	We don't know for sure that none of its internal vertices are in $V(G_i)$. We only know this about x_1, because $x_1 = y$ was chosen for this purpose.

To fix this, let $j > 0$ be the first positive number such that $x_j \in V(G_i)$. (Since $x_l = z$ and $z \in V(G_i)$, we know that j exists, but it's possible that we return to $V(G_i)$ before reaching z; in that case, $j < k$.) Now the truncated walk $(x_0, x_1, x_2, \ldots, x_j)$ represents an ear we can add to G_i: if we define R_{i+1} to be the path represented by this walk, then R_{i+1} is an ear of $R_1 \cup R_2 \cup \cdots \cup R_{i+1}$.

Question:	How do we know this?
Answer:	All the internal vertices of R_{i+1} are not in G_i, so in $G_i \cup R_{i+1}$, they only have two neighbors: their neighbors along the path R_{i+1}. This also shows that the edges R_{i+1} are not in G_i, because they all involve at least one of these internal vertices.

To recap, we've shown that in all cases, if we've picked $R_1, R_2, \ldots, R_i$, then we can continue by picking R_{i+1} that can be part of the ear decomposition. At each step, we add at least one edge that wasn't in the partial ear decomposition yet; therefore we're guaranteed to eventually build all of G, and obtain an ear decomposition of G. □

An important observation is that no matter how we've constructed $R_1, R_2, \ldots, R_i$, if we haven't finished, then we can always choose R_{i+1} somehow. This makes the algorithm in the proof a greedy one! The initial cycle R_1 can be any cycle, and later on, each ear R_i can be any ear. We don't have to worry about making choices that lead us to a dead end later.

In particular, the strategy of picking vertices x, y, z, finding a $y - z$ path, and seeing where it returns to $V(G_i)$ is just one way to guarantee that we find an ear. We do not have to follow this strategy if have an alternative. If we're working with a small graph, such as the cube graph whose ear decomposition we found in Figure 25.4, then it's much easier to just look at the graph and find an ear among the vertices and edges we haven't included yet.

Question:	Is there a shorter ear decomposition of the cube graph?
Answer:	Yes and no. We can work to get to a spanning subgraph more quickly, at which point we just have edges to add; for example, we could let R_1 be a Hamilton cycle in the cube graph. However, we'll have to add those edges one at a time, and we'll still have the same total number of steps.

In a practice problem at the end of this chapter, I'll ask you to prove that this is always true: two ear decompositions of the same graph always have the same number of pieces.

25.5 Induction on ears

Theorem 25.4 can be thought of as an inductive definition of 2-connected graphs: a graph G is 2-connected if and only if it is either a cycle, or the result of adding an ear to a smaller 2-connected graph. Whenever we have such a definition of a combinatorial object of any kind, it suggests that induction is a good strategy for proving properties of such an object. (See Appendix B for more details about such definitions.)

Let me begin by giving you an example, which is an application of 2-connectedness to planar graphs. (I will be lighter on the geometric details, because they are not the focus here. In a more careful approach, using Lemma 21.2 to confirm that all edges must separate two faces is a key step.)

Proposition 25.5. *If a planar graph G is 2-connected, then in every plane embedding of G, the boundary of every face is a single cycle.*

Question:	What are all the ways in which the boundary of a face might fail to be a cycle?
Answer:	It's possible that a face has multiple boundary walks. This happens when G is not connected, and the face touches multiple components of G. It's possible that a boundary walk does not represent a cycle because repeats an edge; we know that this happens when the edge is a bridge. Finally, a boundary walk might not repeat any edges, but still repeat a vertex x. In this case, x is a cut vertex; intuitively, because we can draw a closed curve in the face that only crosses the embedding at x, and separates two parts of G. In a plane embedding of $G - x$, the same closed curve will separate two connected components.

Proof of Proposition 25.5. Since G is planar, begin by picking a plane embedding of G.

Since G is 2-connected, let $R_1, R_2, \ldots, R_k$ be an ear decomposition of G (which exists by Theorem 25.4) and let $G_i = R_1 \cup R_2 \cup \cdots \cup R_i$. Since G_i is a subgraph of G, the plane embedding of G contains a plane embedding of G_i: just erase every vertex and edge of G that's not in G_i.

We will prove by induction on i that in this plane embedding of G_i, the boundary of every face is a single cycle. The base case is $i = 1$. In this case, $G_1 = R_1$ is a cycle ; in every plane embedding of the cycle, there are two faces, an inner face and an outer face, and the entire cycle is the boundary between them.

Now assume for some $i > 1$ that in the plane embedding of G_{i-1}, the boundary of every face is a single cycle. Then G_i is obtained from G_{i-1} by adding the ear R_i: a path whose ends are in G_{i-1}, but whose internal vertices are not. In the plane embedding, this path cannot cross any

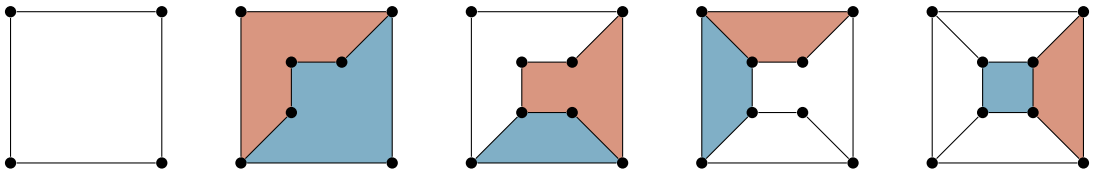


Figure 25.5: In a plane embedding of the cube graph, each ear divides a face in two

edges of G_{i-1}, so there must be some face F in G_{i-1} that contains it entirely. The boundary of F in G_{i-1} is a cycle, by the induction hypothesis.

In G_i, F is divided by R_i into two faces $F_1 \cup F_2$. Their boundaries must share the path R_i, and divide up the original boundary of F somehow. The only way to do this is by two cycles, each of which follows the boundary of F from one endpoint of R_i to the other, and returns along R_i. See Figure 25.5 for an illustration of such induction steps in a planar ear decomposition of the cube graph; in each diagram, the two faces divided by the recently added ear are shaded.

This completes the induction. Now, Proposition 25.5 follows from the $i = k$ case, because G_k is all of G. □

In general, inducting on ear decompositions follows the pattern of this proof. First, we choose an ear decomposition, and verify that the claim holds for R_1 (a cycle). Next, we verify that if the claim is true for G_{i-1}, it holds for G_i: in other words, that the claim continues to hold if we add an ear.

Earlier, we observed that an ear decomposition can be found greedily; in particular, that we can begin the ear decomposition by choosing R_1 to be any cycle we like. This is occasionally useful when writing a proof by induction, since we can make sure that some key vertices end up in R_1.

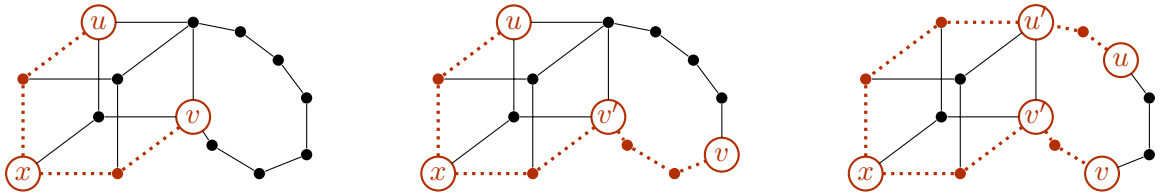
Of course, if we want to do this, we need to prove that the cycle we want exists, first. Once we've shown Theorem 25.1, it will be an excellent tool for the base case, because it's all about cycles existing. Unfortunately, my plan for Theorem 25.1 is to use an induction on ear decompositions! So we will need to begin with a silly lemma that will no longer be necessary once Theorem 25.1 is there to replace it, but will be useful in that particular proof.

Lemma 25.6. *If G is 2-connected, and x is any vertex, then G has a cycle containing x.*

Proof. To have any hope of finding a cycle through x, we need to know that x has two neighbors. Well, if x had only one neighbor, we'd be able to delete that neighbor to disconnect x from the rest of the graph: the neighbor of x would be a cut vertex.

So we can choose two neighbors of x; call them u and v.

Because G is 2-connected, $G - x$ is still connected, and in particular there is a $u - v$ path P in $G - x$. We obtain a cycle through x from P by adding vertex x and edges $\{ux, vx\}$ to P. □



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

Figure 25.6: Three cases in the induction step of Lemma 25.7

Question:	Theorem 25.1 would be false if we considered K_2 to be a 2-connected graph, and so would Lemma 25.6. At which step would the proof fail, if $G = K_2$?
Answer:	When we talk about a vertex deletion that would "disconnect x from the rest of the graph", we assume that there is a rest of the graph, which is false in the case of K_2.

In the next section, we will see two more examples of induction on ear decompositions, and we will see how Lemma 25.6 fits in.

25.6 Finding common cycles

Before we prove Theorem 25.4, we will prove a second lemma. Unlike Lemma 25.6, this one remains useful in the study of 2-connected graphs; it's not just a tool for this one particular proof.

Lemma 25.7. *If G is 2-connected, and u, v, x are any three vertices, then G contains a $u - v$ path that passes through x.*

Proof. By Lemma 25.6, G contains a cycle containing x. Let $R_1, R_2, \ldots, R_k$ be an ear decomposition of G in which R_1 is a cycle containing x, and let $G_i = R_1 \cup R_2 \cup \cdots \cup R_i$. We will prove by induction on i that if u and v are any two vertices in G_i, then G_i contains a $u - v$ path that passes through x.

When $k = 1$, the graph G_1 consists only of the ear R_1: it is a cycle. No matter where u and v are on the cycle, there are two $u - v$ paths going around the cycle from u in either direction, and one of them passes through x, proving the base case.

Now assume that for some $i > 1$ that the paths we want exist in G_{i-1}. Then G_i is obtained from G_{i-1} by adding the ear R_i. Let u and v be arbitrary vertices of G_i; we consider three cases for where they might be, illustrated in Figure 25.6.

Case 1. Both u and v are vertices of G_{i-1}. By the induction hypothesis, G_{i-1} contains a $u - v$ path through x. This is also a path in G_i.

Case 2. Only one of u and v is a vertex of G_{i-1}; without loss of generality, $u \in V(G_{i-1})$ and v is an internal vertex of R_i. In this case, let u' be the start or end of R_i. By the induction

hypothesis, G_{i-1} contains a $u' - v$ path through x. We can extend it to a $u - v$ path through x in G_i if we begin by going from u to u' along the ear R_i.

Case 3. Both u and v are internal vertices of R_i. In this case, let u' and v' be the ends of R_i closer to u and to v, respectively; that is, R_i is a path of the form $(u', \dots, u, \dots, v, \dots, v')$. By the induction hypothesis, G_{i-1} contains a $u' - v'$ path through x. We can extend it to a $u - v$ path through x in G_i if we begin by going from u to u' and end by going from v to v', both along the ear R_i.

In all cases, we've found the $u - v$ path through x we wanted; therefore G_i contains such a path for any $u, v \in V(G_i)$. By induction, this is true for all i and in particular for $i = k$; because $G_k = G$, this proves the lemma. □

With the aid of Lemma 25.3, a second induction on the ear decomposition is now enough to prove Theorem 25.1 that any two vertices x and y of a 2-connected graph G lie on a common cycle.

Proof of Theorem 25.1. Let x and y be two vertices of a 2-connected graph G; we want to show that there is a cycle in G containing both x and y.

By Lemma 25.6, G contains a cycle containing x. Let $R_1, R_2, \dots, R_k$ be an ear decomposition of G in which R_1 is a cycle containing x, and let $G_i = R_1 \cup R_2 \cup \cdots \cup R_i$. We will prove by induction on i that if $y \in V(G_i)$, then G_i contains a cycle that passes through both x and y.

When $i = 1$, this is automatic: $G_1 = R_1$ is the cycle we want.

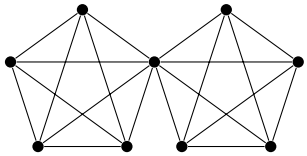
Now assume for some $i > 1$ that every vertex $y \in V(G_{i-1})$ lies on a cycle in G_{i-1} that also passes through x. We would like to prove the same thing for G_i, which is obtained from G_{i-1} by adding the ear R_i. Since we have the induction hypothesis, we only need to find a cycle through x and y when y is one of the new vertices: an internal vertex of R_i.

Let u and v be the ends of ear R_i. By Lemma 25.6, there is a $u - v$ path P in G_{i-1} passing through x. The paths P and R_i have only their ends u and v in common: none of the internal vertices of R_i are in $V(G_{i-1})$, and all vertices of P are. Therefore the union $P \cup R_i$ is a cycle containing x (because P contains x) and y (because R_i contains y).

By induction, for every $y \in V(G_k)$ there is a cycle in G_k that passes through x and y; since $G = G_k$, this proves the theorem. □

25.7 Practice problems

1. Give an example of each of the following:
   a) A 10-vertex graph with 8 cut vertices.
   b) A graph with the same number of vertices and edges as in (a), but only one cut vertex.
   c) A regular graph with only one cut vertex.
2. Let G be a graph consisting of two copies of K_5 joined at a vertex:



How many spanning trees does G have?

3. Suppose G has a cut vertex x, and that the graphs $H_1, H_2, \ldots, H_k$ are the subgraphs of G meeting at x (as in Figure 25.2).

   a) Explain why knowing the independence numbers $\alpha(H_1), \alpha(H_2), \ldots, \alpha(H_k)$ is not enough to find $\alpha(G)$.

   b) Without knowing G, if all I tell you is the values of $\alpha(H_1), \alpha(H_2), \ldots, \alpha(H_k)$, what are the minimum and maximum possible values of $\alpha(G)$?

4. Prove that if G is 2-connected, and H is a *subdivision* of G (as defined in Chapter 22), then H is 2-connected.

5. Find an ear decomposition of:

   a) $K_{3,3}$.

   b) $K_{2,5}$.

   c) $K_{2,n}$ for every n.

6. You might wonder: is there a shortest ear decomposition in a graph? In fact, every ear decomposition contains the same number of pieces.

   a) Suppose that G has the ear decomposition $R_1, R_2, \ldots, R_k$, where R_1 is a cycle of length l_1 and for each $i \geq 2$, R_i is a path of length l_i.

   Find the number of edges in G in terms of $l_1, l_2, \ldots, l_k$.

   b) Find the number of vertices in G in terms of $l_1, l_2, \ldots, l_k$.

   c) If G has n vertices, m edges, and k ears in an ear decomposition, find a relationship between n, m, and k from your answers to the first two parts.

   Conclude that the value of k is predetermined by G, and doesn't depend on the ear decomposition.

7. One way to interpret Theorem 25.1 is that for any two vertices x, y in a 2-connected graph G, there are two $x-y$ paths that share none of their internal vertices. (Such paths are said to be *internally disjoint*, a term we will introduce in Chapter 26.) Here is an INCORRECT proof of a FALSE generalization:

   Claim. If G is 2-connected and P is any $x-y$ path, there is a second $x-y$ path P' that shares no vertices with P other than x and y.

   "Proof". Suppose there is no such path P'. That means that all other paths P' end up sharing some other vertex of P. But then, deleting that vertex of P destroys all $x-y$ paths, and so that vertex was a cut vertex. This cannot happen in a 2-connected graph; contradiction! So a path P' must exist.

a) Point out the mistake in the proof.

b) Give a counterexample to the claim.

8. Let G be a connected graph with at least 3 vertices in which xy is a bridge (that is, $G-xy$ is not connected.)

   a) Prove that either x or y is a cut vertex of G. Give an example showing that they're not necessarily both cut vertices of G.

   b) Prove that xy is a cut vertex of the line graph $L(G)$.

9. In a 3-regular graph:

   a) Prove that a vertex x is a cut vertex if and only if it is incident to a bridge.

   b) Prove that every cut vertex is incident to either 1 or 3 bridges.

   c) Prove that the number of cut vertices is even.

10. The ear decompositions in this chapter are sometimes called proper ear decompositions. There is also a corresponding notion of improper ear decompositions. In the improper case, an ear of a graph G is also allowed[2] to be a cycle in which every vertex except one has degree 2. (That is, when we add an ear, we are allowed to start and end at the same vertex.)

    a) Prove by example that a graph with an improper ear decomposition is not necessarily 2-connected.

    Instead, improper ear decompositions correspond to *2-edge-connected* graphs: connected graphs with no bridges. (These are a special case of a definition we will make in Chapter 26.)

    b) Prove that if a graph G has an improper ear decomposition, then it is 2-edge-connected.

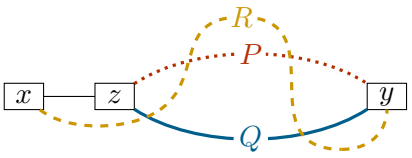
    c) Prove that if a graph is 2-edge-connected, then it has an improper ear decomposition.

11. There is an alternate proof of Theorem 25.1 which does not use ear decompositions. Instead, we prove that any two vertices x, y lie on a common cycle (equivalently, there are two $x-y$ paths with no internal vertices in common) by inducting on the distance $d(x, y)$.

    a) For the base case, prove (without relying on any of our other results) that if G is 2-connected, then for any edge $xy \in E(G)$, there is a cycle containing xy.

    b) For the induction step, we assume that Theorem 25.1 holds for any two vertices at some distance $k \geq 1$, and let x and y be two vertices with $d(x, y) = k + 1$. To apply the induction hypothesis, we let z be the first vertex on a shortest $x - y$ path, so that $d(z, y) = k$. Let P and Q be two internally disjoint $z - y$ paths, and let R be an $x - y$ path in $G - z$, as shown below:

[2] An "improper" ear decomposition should maybe be called a "not-necessarily-proper" ear decomposition: it could be proper, it just doesn't have to be.



Prove that no matter how R intersects P and Q, we can find two $x - y$ paths with no internal vertices in common.

26 Connectivity

The purpose of this chapter

I'm hoping that the idea behind the definitions in this chapter feels intuitive: that to measure the resilience of a graph, it makes sense to see how much you have to remove from the graph to disconnect it. There is a lot more to these definitions, and they turn out to have connections to all parts of graph theory. However, it's going to take a bit of time to get there, and the content of this chapter is mostly setup.

When I've taught this material in the past, I've sometimes focused only on vertex connectivity and not on edge connectivity, for reasons of time. If you do have the time, I think it is nice to look at both, because the contrast helps you understand the logic of each case. For example, if it's hard at first to see why internally disjoint paths are what we need for lower bounds on $\kappa(s,t)$, then edge-disjoint paths and their lower bounds on $\kappa'(s,t)$ provide a second example of the same idea for comparison.

The final section on duality is a topic I could have discussed earlier in the book, but now we have lots of examples to look back to. I should note that in graph theory, duality is a relationship between optimization problems that just sometimes appears; we will occasionally prove it appears, but not explain why. Using linear programming, we could obtain more of an explanation, and even deduce what the correct duals of some optimization problems should be without seeing them before. That would take us too far afield, though.

26.1 Vertex and edge cuts

In the previous chapter, we defined cut vertices and 2-connected graphs; much earlier in the book, we defined bridges. In this chapter, we will discuss generalizations of all three definitions.

We begin with the definition of a **vertex cut** in a graph G. This is subset $U \subseteq V(G)$ such that when all vertices in U are deleted, the remaining graph $G - U$ is not connected.

Don't confuse "vertex cut" and "cut vertex", though they have the same words in a different order! A cut vertex is a vertex, with "cut" as the adjective: it is a vertex which cuts. A vertex cut is a cut (as in, "I sliced the cake in half with a single cut"), with "vertex" as the adjective: it is a cut made up of vertices.

The two terms are related, though. If G is a connected graph, a vertex x is a cut vertex of G if and only if the set $\{x\}$ is a vertex cut of G.

Question:	What if G is not connected?
Answer:	In this case, cut vertices are those that increase the number of connected components. Meanwhile, $\{x\}$ is almost always a vertex cut; the exception is when x is an isolated vertex and the rest of the graph is connected.

Usually, this will not matter, because we will be looking for the smallest vertex cuts we can. If G is not connected, we will simply say that $\varnothing$ (the empty set) is a vertex cut.

Making vertex cuts an optimization problem in this way makes sense, because it is not surprising when we disconnect a graph by deleting many vertices. We ask: what is the least number of vertices whose deletion disconnects the graph? Unfortunately, before we make that into a definition, we have to tackle an awkward obstacle.

Question:	Is it always possible to disconnect a graph by deleting enough vertices?
Answer:	Almost always, except for complete graphs! In K_n, any two vertices are adjacent, and will remain adjacent no matter how many other vertices we remove. This means that (aside from the question of what happens if we delete every single vertex) we can't disconnect K_n by deleting some of its vertices: K_n has no vertex cuts.

This is related to what happened with 2-connected graphs in the previous chapter: we didn't want K_2 to be 2-connected, because most of the theorems about 2-connected graphs do not apply to it, so we specifically excluded it from the definition.

Accordingly, we define the **connectivity** (or **vertex connectivity**) $\kappa(G)$ of a graph G to be the smallest number of vertices in a vertex cut of G, if such a vertex cut exists. If not, then G is isomorphic to K_n for some n, and we "artificially" define $\kappa(K_n) = n - 1$.

We also say that a graph G is ***k*-connected** if $\kappa(G) \geq k$. This is in line with our previous definitions. For graphs with at least 2 vertices, "1-connected" and "connected" are synonyms; meanwhile, 2-connected graphs (as defined in Chapter 25) still have the same definition if we set $k = 2$ in the definition of k-connected graph.

Question:	Why does the definition of k-connected graphs have an inequality: $\kappa(G) \geq k$, rather than $\kappa(G) = k$?
Answer:	We say "G is k-connected" when we want to say that G is connected enough for something to work. For example, all 2-connected graphs have ear decompositions, even if they are also 3-connected or 100-connected. This is similar to the way we defined k-colorable graphs: they are the graphs with $\chi(G) \leq k$.

We must specify "vertex cut" and we often specify "vertex connectivity" because all of these definitions have an equivalent for deleting edges, rather than vertices, of a graph. The story is almost the same for edges, so I will just present all the definitions at once.

An **edge cut** in a graph G is a subset $X \subseteq E(G)$ such that when all edges in X are deleted, the remaining graph $G - X$ is not connected.

The **edge connectivity** of G, denoted $\kappa'(G)$, is the smallest number of edges in an edge cut of G, if G has at least two vertices. If G has only one vertex, then we set $\kappa'(G) = 0$.

Finally, a graph G is ***k*-edge-connected** if $\kappa'(G) \geq k$.

Question:	Do we need to take any special care when $G = K_n$?
Answer:	Not unless $n = 1$ (which the definition addresses). When G at least 2 vertices, it is always possible to disconnect G by deleting some edges: in particular, deleting all edges would be enough.

In Chapter 20, we discussed the correspondence between "vertex versions" and "edge versions" of a problem, and used the same notation: for example, $\alpha'(G)$ and $\chi'(G)$ are the edge versions of $\alpha(G)$ and $\chi(G)$. I will use the same notation here. In other sources, $\lambda(G)$ is also a common notation for edge connectivity, but I'd rather not make you memorize even more Greek letters.

Question:	Often the edge version of a problem is just the vertex version, but applied to the line graph $L(G)$ rather than G. Is that true here—is $\kappa'(G) = \kappa(L(G))$?
Answer:	Not quite. I leave the details to practice problem, but a relationship exists only in one direction: $\kappa'(G) \leq \kappa(L(G))$.

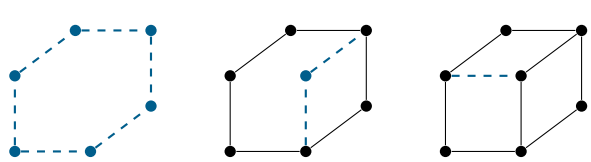
Let X be an edge cut in G, and let S be the set of vertices in one connected component of $G-X$. Then X must contain all edges of G with one endpoint in S and one endpoint in $V(G) - S$. Though X could contain other edges, this is wasteful: just deleting all the edges between S and $V(G) - S$ is enough to disconnect G.

Accordingly, for $S \subseteq V(G)$, we define the **edge boundary** of S to be the set of all edges of G with exactly one endpoint in S. We denote it $\partial_G(S)$ or just $\partial(S)$ when the graph G does not need to be specified.

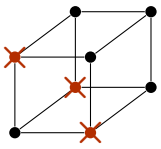
Suppose X is a minimum edge cut, or even a *minimal* one (by the distinction described in Chapter 13): we cannot remove any edges from X and still have an edge cut. In that case, $X = \partial(S)$ for some S. Another way to phrase this is the following proposition:

Proposition 26.1. *Every edge cut X in a connected graph contains an edge boundary $\partial_G(S)$ for some set of vertices S that is neither $\varnothing$ nor all of $V(G)$. In particular, $\kappa'(G)$ can be computed as the minimum size $|\partial_G(S)|$ of all such edge boundaries.*

We do not consider $S = \varnothing$ or $S = V(G)$ because these are the two cases that can never be a connected component of $G - X$, when X is an edge cut.



(a) An ear decomposition of $Q_3 - x$



(b) A vertex cut in Q_3

Figure 26.1: Determining the connectivity of the cube graph

Question:	Is every set of the form $\partial(S)$ a minimal edge cut?
Answer:	No: it's possible that $\partial(S)$ contains a smaller edge cut of the form $\partial(T)$ for a different set T. For an extreme example, consider a bipartite graph with bipartition (A, B). Then the edges $\partial(A)$ we need to separate A from B are all the edges of G; the set $\partial(\{x\})$ for a single vertex x can be much smaller.

26.2 Some examples

In Chapter 25, we saw that the cube graph Q_3 is 2-connected. In fact, we can go one step further.

Proposition 26.2. $\kappa(Q_3) = 3$.

Proof. Ear decompositions are a nice way to prove that graphs are 2-connected, but we don't yet have an equally nice way to show that graphs are 3-connected. We will find one later in this chapter, but for now, let's resort to trickery.

We know Q_3 has no cut vertex; does it have a cut $\{x, y\}$ of size 2? If it did, then in $Q_3 - x$, vertex y would be a cut vertex: deleting it would bring us to $Q_3 - \{x, y\}$, which by assumption is not connected. So we can check for this possibility by checking whether $Q_3 - x$ is 2-connected.

Normally, we'd have to do this by checking 8 possibilities for x. In the case of the cube graph, there are enough symmetries (automorphisms of the cube graph) that it's enough to check one vertex: all 8 cases will be identical.

Figure 26.1a shows an ear decomposition of $Q_3 - x$ for an arbitrary vertex x. (Well, it's only arbitrary mathematically; I deleted the vertex "in the back" of the cube to make the diagram look nicer.) This shows that $Q_3 - x$ is 2-connected, so Q_3 is 3-connected.

Can it be 4-connected? No, because Q_3 has a 3-vertex cut, shown in Figure 26.1b. Therefore $\kappa(Q_3)$ is exactly 3. □

The vertex cut we chose in Figure 26.1b is rather boring; we've disconnected a single vertex from the rest of the graph. Such a vertex cut is possible in every graph G, except in the unfortunate case of (you guessed it) complete graphs. By deleting all edges incident to a vertex x, we also disconnect x from the rest of G, so there is an edge cut of this type as well. To make the cuts as small as possible, we want to choose a vertex x of degree $\delta(G)$: the minimum degree of G.

This is not universal; we wouldn't study connectivity if it always turned out to be equal to the minimum degree. However, the following relationship (also proved by Whitney, like most of the results in Chapter 25) always holds:

Theorem 26.3. *For any graph G, $\kappa(G) \leq \kappa'(G) \leq \delta(G)$.*

Proof. The second inequality, $\kappa'(G) \leq \delta(G)$, follows from our discussion just before the theorem: if x is a vertex of degree $\delta(G)$, then deleting all $\delta(G)$ edges incident to x will disconnect x from the rest of the graph, so it is always an edge cut. (As usual with optimization problems, finding a particular solution gives us an inequality, because there might be better solutions.)

The more difficult inequality is the first. To prove that $\kappa(G) \leq \kappa'(G)$, we need to do the following: given an edge cut X of size $\kappa'(G)$, find a vertex cut U with $|U| \leq |X|$. By Proposition 26.1, we can assume that $X = \partial_G(S)$ for some set S such that $\varnothing \neq S \subsetneq V(G)$.

I like the proof of this inequality because it is an excellent exercise in attempting an argument, identifying the holes in it, and then patching the holes. A few rounds of this, and we will have a complete proof.

Let's begin with the following strategy: to find U, go through the edges in $\partial(S)$, and for each $xy \in \partial(S)$, add either x or y to U. Deleting an endpoint of xy also deletes the edge xy, so after these deletions, S (or what's left of it) is disconnected from the rest of the graph.

Question:	What could go wrong?
Answer:	It's possible that in deleting these endpoints, we've actually deleted every vertex in S, or every vertex not in S, so the remaining graph is still connected.

Okay, so let's make sure that we avoid this. Begin by picking a vertex $s \in S$ and a vertex $t \notin S$. For every edge $xy \in \partial(S)$, add either x or y to U, but with a restriction: if $x = s$, add y to U, and if $y = t$, add x to U. Then in $G - U$, there is no way to get from s to t, because $s \in S$, $t \notin S$, and we have destroyed all edges leaving S.

Question:	What could go wrong?
Answer:	The restriction we've added doesn't leave us an option to choose if one of the edges in $\partial(S)$ is the edge st.

Okay, fair enough; let's make sure that we pick a vertex $s \in S$ and a vertex $t \notin S$ that are not adjacent.

Question:	What could go wrong?
Answer:	Maybe every vertex in S is adjacent to every vertex not in S, so that no such s and t can be picked.

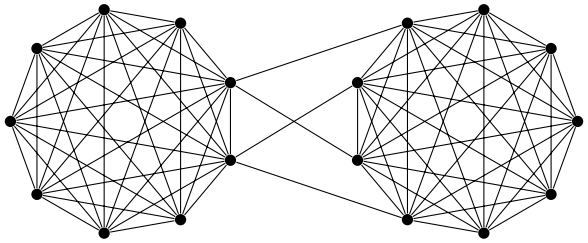


Figure 26.2: A graph with $\kappa(G) = 2$, $\kappa'(G) = 4$, and $\delta(G) = 8$

This is a surprising situation to be in! Let G have n vertices, and let $|S| = k$. Then $\partial(S)$ contains all edges between the k vertices in S and the $n - k$ vertices not in S: this is a lot: $|\partial(S)| = k(n-k)$. We have assumed that $\kappa'(G) = |\partial(S)| = k(n-k)$, but in fact, $k(n-k) > n-1$ except when $k = 1$ or $k = n - 1$.

Since we already know that $\kappa'(G) \leq \delta(G)$, and in every n-vertex graph, $\delta(G) \leq n - 1$, only one possibility remains: $\kappa'(G) = n - 1 = \delta(G)$.

Question:	What graph must G be?
Answer:	G must be isomorphic to K_n; this is the only n-vertex graph with minimum degree $n - 1$.

When G is isomorphic to K_n, $\kappa'(G) = \delta(G) = n - 1$, we've defined $\kappa(G) = n - 1$ artificially (and this theorem is part of the reason why). So the inequality $\kappa(G) \leq \kappa'(G)$ is satisfied.

Now we just have to carry out our argument assuming G is not a copy of K_n. This means that $\delta(G) < n - 1$, so $\kappa'(G) < n - 1$, so $|\partial(S)| < n - 1$. This means that not all $k(n - k)$ of the possible edges with one endpoint in S are actually present. Therefore we can choose $s \in S$ and $t \notin S$ which are not adjacent, and construct U by the following rule: for each edge $xy \in \partial(S)$, add either x or y to U, taking care not to add either s or t. As we've already shown, this is a vertex cut of size at most $|\partial(S)|$, proving that $\kappa(G) \leq \kappa'(G)$. □

Since this is a section called "Examples", let me reassure you by means of an example that apart from the constraint $\kappa(G) \leq \kappa'(G) \leq \delta(G)$, almost all triples $(\kappa(G), \kappa'(G), \delta(G))$ are possible. There is only one other constraint on such triples: if $\kappa(G) = 0$, it is because G is not connected, so $\kappa'(G) = 0$ as well.

Let $1 \leq a \leq b \leq c$. To find a graph where $\kappa(G) = a$, $\kappa'(G) = b$, and $\delta(G) = c$, begin with two copies of K_{c+1}. Let A be a set of a vertices in the first copy and let B be a set of b vertices in the second copy; add b edges between A and B, making sure to use each vertex in A at least once and each vertex in B exactly once. Call the result $G_{a,b,c,d}$. An example of this construction with $a = 2$, $b = 4$, and $c = 8$ is shown in Figure 26.2.

Question:	Why is $\kappa(G_{a,b,c,d}) = a$?
Answer:	The vertices in A are an a-vertex cut. If we delete fewer than a vertices, we can find vertices $x \in A$ and $y \in B$ that are adjacent, and have not been deleted; since every vertex of $G_{a,b,c,d}$ is adjacent to either x or y, the graph will still be connected.

Question:	Why is $\kappa'(G_{a,b,c,d}) = b$?
Answer:	The edges between A and B are a b-edge cut. If we delete fewer than b edges, then in particular (because $b \leq c$, and $\kappa'(K_{c+1}) = c$) each copy of K_{c+1} is connected; also, at least one edge between the two copies of K_{c+1} remains, connecting them together.

Question:	Why is $\delta(G_{a,b,c,d}) = c$?
Answer:	We started with two copies of K_{c+1}, in which every vertex has degree c. We added edges to $a + b$ vertices, which is at most $2c$; therefore at least two vertices still have degree c.

Question:	Why is there even a d in $G_{a,b,c,d}$?
Answer:	No reason; I'm just messing with you.

26.3 Local connectivity

We often want to solve a problem which is similar to finding a vertex cut or edge cut in a graph, but more specialized.

As I'm writing this chapter, it is November, and in the United States, Thanksgiving is coming up. Many people travel long distances to celebrate Thanksgiving with their family, but it's also almost winter, so if you take a plane across the US, you might have to deal with cancellations due to snow and other winter weather. Some flights might be canceled, deleting an edge in the graph of possible airplane trips. In extreme circumstances, an entire airport might shut down and divert all flights, deleting a vertex.

How many of these events must happen for you to be entirely unable to get to your destination? This is a bit like the question of computing edge connectivity (in the case of flight cancellations) or vertex connectivity (in the case of closed airports).

In practice, vertex and edge connectivity of this graph are low. I can speak to this from personal experience: I have spent three years living in Urbana, Illinois. The local airport has flights to only two other airports: Chicago O'Hare and Dallas–Fort Worth. It was not a very reliable means of travel, especially in the winter!

Suppose, however, that you live in New York City and your family lives in Los Angeles. You do not care if a few cancellations are enough to disconnect you from Urbana, because you're

not going to Urbana. In your case, the number of cancellations that would have to occur for all routes you could take to be blocked off is truly implausible.

To model situations like this, we need a more specialized definition. Here, let s and t be two vertices in a graph G; we will specifically consider what it takes to separate s from t.

- An **$s-t$ vertex cut** (or just **$s-t$ cut**) in G is a vertex cut U that separates s from t: vertices s and t are in different connected components of $G-U$. (In particular, U must not contain either s or t.)

  The **$s-t$ connectivity in G**, denoted $\kappa_G(s,t)$ or just $\kappa(s,t)$ if there is no need to specify the graph, is the smallest number of vertices in an $s-t$ cut.

- An **$s-t$ edge cut** in G is an edge cut X that separates s from t: vertices s and t are in different connected components of $G-X$.

  The **$s-t$ edge connectivity in G**, denoted $\kappa'_G(s,t)$ or just $\kappa(s,t)$, is the smallest number of edges in an $s-t$ edge cut.

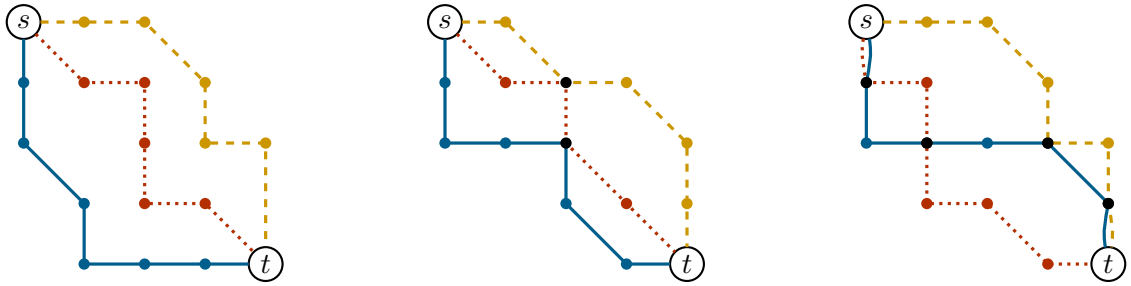
Question:	What happens to $\kappa(s,t)$ if s and t are adjacent?
Answer:	In this case, there is no vertex cut that separates s from t; we either disregard this case entirely or set $\kappa(s,t)=\infty$. However, there are no problems with defining $\kappa'(s,t)$; we just have to make sure that st is one of the edges we delete.

There are, broadly speaking, two reasons to care about $s-t$ cuts of the two types. One reason is an application such as your hypothetical Thanksgiving travel plans from New York City to Los Angeles, where you really only care about whether an $s-t$ path survives. There is another: as we will discover in the next few chapters, finding $\kappa_G(s,t)$ is sometimes much easier than finding $\kappa(G)$. If we really want to know $\kappa(G)$, we might perform several computations of $\kappa_G(s,t)$ instead.

Question:	If you were given a table of $\kappa_G(s,t)$ and $\kappa'_G(s,t)$ for all pairs of vertices s and t, could you use it to determine $\kappa(G)$ and $\kappa'(G)$?
Answer:	Yes $\kappa'(G)$ is the minimum of $\kappa'_G(s,t)$ over all pairs $\{s,t\}$, and if G is not a complete graph, $\kappa(G)$ is the minimum of $\kappa_G(s,t)$ over all non-adjacent pairs $\{s,t\}$.

The quantities $\kappa(s,t)$ and $\kappa'(s,t)$ are the solutions to optimization problems: of all $s-t$ cuts of the appropriate type, we want to find the smallest. Finding upper bounds is, in both cases, straightforward (if not necessarily easy): if you find an $s-t$ cut U, then you know that $\kappa(s,t)\leq|U|$. But how do we prove a lower bound?

Here's one possible answer. Suppose that the graph contains a subgraph such as the one in Figure 26.3a. It is the union of three $s-t$ paths (a blue path, a red path, and a yellow path). Moreover, the three paths share no internal vertices. Then it's clear that $\kappa(s,t)\geq 3$: we need to delete at least one of the internal vertices from each path if we want to disconnect s from t.



(a) Internally disjoint paths (b) Edge-disjoint paths (c) Just some paths

Figure 26.3: Three different sets of three $s - t$ paths

Question:	Why is it important that the three paths share no internal vertices?
Answer:	Consider the paths in Figure 26.3b, instead. They provide three ways to get from s to t, but some vertices are repeated. It's possible to destroy multiple paths by deleting just one vertex, and in fact all three paths can be destroyed by deleting 2 vertices.

The paths in Figure 26.3b are still useful, though: they share no edges. If we seek to disconnect s from t by deleting edges, the existence of such paths proves that at least three edges need to be deleted: one from each path. Compare this to the paths in Figure 26.3c, which have no special properties relative to each other! Here, we can destroy all three paths by deleting just two edges.

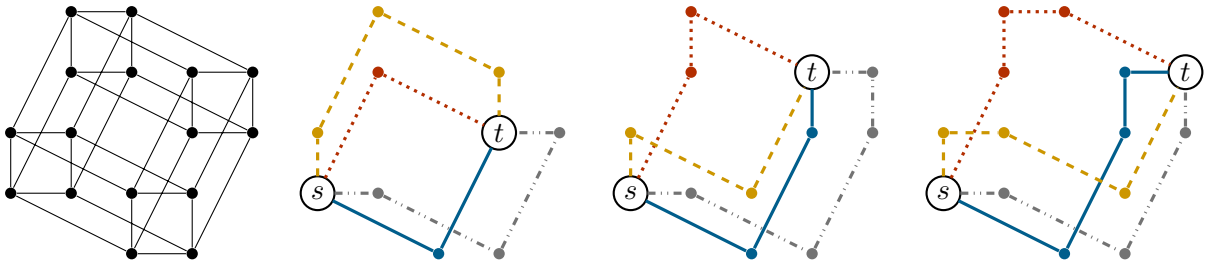
To generalize, we say that two paths in a graph are **internally disjoint** if they do not share any internal vertices; they are **edge-disjoint** if they do not share any edges. I want to emphasize that both of these definitions are properties a pair of paths can have. It would not make sense to look at an $s - t$ path and ask, "Is this $s - t$ path internally disjoint or not?"

For a set of paths, such as what we see in Figure 26.3, the formally correct phrasing would be that the paths in a set are "pairwise internally disjoint" if no two paths share internal vertices, and "pairwise edge-disjoint" if no two paths share edges. The word "pairwise" emphasizes that we consider the paths two at a time, rather than all at once; for a set of 100 paths, it's not enough to know that there's no vertex that all 100 paths pass through.

This is a mouthful. To make it a little bit easier, I will talk about a "set of internally disjoint paths" or a "set of edge-disjoint paths" to mean that any two paths in the set are internally disjoint or edge-disjoint, respectively. In case you are confused, always think back to this example and the argument we are trying to make with the set of paths! We want the paths to be sufficiently separate that to destroy them all (by deleting whichever of vertices or edges we're thinking about) we need to handle each path separately.

We will do a lot more with this idea in the next two chapters. For now, I will use it to go one step further beyond Proposition 26.2.

Proposition 26.4. *The hypercube graph Q_4 is 4-connected.*



(a) Q_4 (b) $\kappa(0000, 0011) \geq 4$ (c) $\kappa(0000, 0111) \geq 4$ (d) $\kappa(0000, 1111) \geq 4$

Figure 26.4: Sets of internally disjoint paths in Q_4

Proof. Our strategy will be to compute $\kappa(Q_4)$ by computing $\kappa(s,t)$ for every s and every t. But first, let me begin with a bit of review of Q_4. This graph has $2^4 = 16$ vertices, which are 4-bit binary strings $b_1b_2b_3b_4$. Two vertices are adjacent if the binary strings agree in 3 out of 4 positions, and only disagree in 1 position.

The hypercube graph Q_4 has many automorphisms. To review some concepts you may not have thought about since Chapter 2, I'll write them out in detail. We'll need two types of automorphisms of Q_4 to reduce the casework we have to do:

- For each position i, let $\varphi_i \colon V(Q_4) \to V(Q_4)$ be the function that flips the i^{th} bit; for example, $\varphi_3(0101) = 0111$. This is a bijection because it is its own inverse. It is an automorphism because when we compare two binary strings x and y, if we flip the i^{th} bit in both of them, it doesn't change which positions they agree in.
- For every two positions i and j, let $\varphi_{ij} \colon V(Q_4) \to V(Q_4)$ be the function that swaps the i^{th} and j^{th} bits: for example, $\varphi_{13}(1001) = 0011$. This is a bijection because it is its own inverse. It is an automorphism because when we compare two binary strings x and y, applying φ_{ij} to both might move a position they disagree in (from i to j or from j to i) but won't change the number of disagreements.

Our goal is to compute $\kappa(s,t)$ for every pair of vertices $\{s,t\}$. There are $\binom{16}{2} = 90$ pairs to test if we do it the hard way. However, by applying the automorphisms above, we can reduce all 90 cases to just a few! The general logic is this: if φ is any automorphism of Q_4, then $\kappa(s,t) = \kappa(\varphi(s), \varphi(t))$.

Question:	How do we know this?
Answer:	If there is a vertex cut U separating s from t, then applying φ to its vertices gives a vertex cut of the same size separating $\varphi(s)$ from $\varphi(t)$. In the other direction, applying φ^{-1} to the vertices of a $\varphi(s) - \varphi(t)$ cut gives an $s - t$ cut of the same size.

Consider any pair $s, t \in V(Q_4)$. First, some composition of the automorphisms $\varphi_1, \ldots, \varphi_4$ maps s to 0000 and t to some vertex t'. Therefore $\kappa(s,t) = \kappa(0000, t')$. The second type of automorphisms (the φ_{ij}'s) leave 0000 unchanged, but some composition of them sorts the bits of t' in ascending order: to one of 0001, 0011, 0111, or 1111. Therefore

$$\kappa(Q_4) = \min\Big\{\kappa(0000, 0001),\ \kappa(0000, 0011),\ \kappa(0000, 0111),\ \kappa(0000, 1111)\Big\}.$$

Of these four cases, $\kappa(0000, 0001) = \infty$ because 0000 and 0001 are adjacent; we can skip this one. For the other three, the vertex cut $\{0001, 0010, 0100, 1000\}$ consisting of all four neighbors of 0000 separates 0000 from the other vertices, so they are all at most 4. Finally, Figure 26.4 shows that in all three of these cases, we can find a set of 4 internally disjoint paths. Therefore 4 is a lower bound as well, and we conclude that $\kappa(Q_4) = 4$. □

Question:	What is $\kappa'(Q_4)$?
Answer:	By Theorem 26.3, it is sandwiched between $\kappa(Q_4)$ and $\delta(Q_4)$. Both of these are 4, so $\kappa'(Q_4) = 4$ as well.

26.4 Duality

The technique of internally disjoint paths seems pretty powerful based on this example, but there is an unresolved question. If we use this technique to prove lower bounds on $\kappa(s, t)$ or $\kappa'(s, t)$, will we always be able to prove a lower bound equal to the true value of these numbers? Or is it possible that in some cases, $\kappa(s, t) = 4$, but the largest set of internally disjoint paths only contains 3 paths?

In the next chapter, we will prove that this will never happen: not for vertices, and not for edges. In preparation, let me tell you about one framework to think about the relationship between $s - t$ cuts and sets of disjoint paths: optimization duality.

Optimization duality is a relationship that can exist between two optimization problems: a maximization problem and a minimization problem. We want to be fairly general here, but not so general that we get lost in abstraction. Let's say that:

- The maximization problem is, given a set $\mathcal{X}$ and a function $f\colon \mathcal{X} \to \mathbb{R}$, to find an element $x \in \mathcal{X}$ such that $f(x)$ is as large as possible. Written concisely, it is the problem of finding $\max\{f(x) : x \in \mathcal{X}\}$.
- The minimization problem is, given a set $\mathcal{Y}$ and a function $g\colon \mathcal{Y} \to \mathbb{R}$, to find an element $y \in \mathcal{Y}$ such that $g(y)$ is as small as possible. Written concisely, it is the problem of finding $\min\{g(y) : y \in \mathcal{Y}\}$.

Let's try putting some problems we've already studied into this language. Don't worry too much about how the functions f and g are implemented; it's enough to know they exist.

Question:	In the maximization problem of finding the matching number of a graph G, what is $\mathcal{X}$ and what is f?
Answer:	The set $\mathcal{X}$ is the set of all matchings in G. If $x \in \mathcal{X}$ is such a matching, then $f(x)$ is the number of edges in it.

Question:	In the minimization problem of finding the chromatic number of a graph G, what is $\mathcal{Y}$ and what is g?
Answer:	The set $\mathcal{Y}$ is the set of all proper colorings of G. If $y \in \mathcal{Y}$ is such a coloring, then $g(y)$ is the number of colors used by y.

Optimization duality comes in two forms: weak and strong. To begin with, we say that a maximization problem $\max\{f(x) : x \in \mathcal{X}\}$ and a minimization problem $\min\{g(y) : y \in \mathcal{Y}\}$ are in *weak duality* if $f(x) \geq g(y)$ for every $x \in \mathcal{X}$ and every $y \in \mathcal{Y}$. Equivalently,

$$\max\{f(x) : x \in \mathcal{X}\} \leq \min\{g(y) : y \in \mathcal{Y}\}$$

is the inequality that characterizes weak duality.

Why is this helpful? Well, consider the example of matchings and vertex covers in graphs: in the entire textbook, this is perhaps our best example of dual optimization problems. By Proposition 13.4, whenever M is a matching and U is a vertex cover in the same graph G, we have $|E(M)| \leq |U|$: the maximization problem is always bounded above by the minimization problem. This means we can try to find small vertex covers to get upper bounds on the size of a matching: upper bounds that are otherwise hard to come by.

In general, duality fills a gap in our understanding of an optimization problem. By default, if we have an optimization problem to maximize $f(x)$ over all $x \in X$, and we've worked on it for a while, we might not have anything to show for our progress except some element $x^* \in X$ for which $f(x^*)$ is the largest found so far. This proves a lower bound on the optimal value: $\max\{f(x) : x \in \mathcal{X}\}$ is at least $f(x^*)$.

To prove an upper bound on $\max\{f(x) : x \in \mathcal{X}\}$, we could try every element of $\mathcal{X}$ to see which is best. Or, we could come up with and prove an insightful theorem that gives a general upper bound. These both seem very hard. But if we have a dual problem $\min\{g(y) : y \in \mathcal{Y}\}$, then we can switch to working on that problem, instead! After a while, the element $y^* \in Y$ for which $g(y^*)$ is the smallest found so far is probably pretty good. This value $g(y^*)$ will be an upper bound on our original problem: by weak duality, $\max\{f(x) : x \in \mathcal{X}\}$ is at most $g(y^*)$.

The reason weak duality is "weak" is that the bounds we get this way might turn out to be very bad. Even if we've found the best x^* we can, weak duality does not guarantee us a way to ever prove that it's the best. If the two dual problems meet in the middle, and

$$\max\{f(x) : x \in \mathcal{X}\} \leq \min\{g(y) : y \in \mathcal{Y}\},$$

we say that they are in *strong duality*.

In the worst of all possible worlds, there is no structure to $\mathcal{X}$, no meaning to f, and no way to find $\max\{f(x) : x \in \mathcal{X}\}$ except by exhaustive search. In such a world, even if you find the optimal solution x^* after hundreds of hours of computing time, skeptics won't trust you: they will say that maybe your software had a bug in it, and you missed some cases. But with strong duality,[3] you can find an optimal solution y^* to the dual problem. Now you can silence the skeptics by simply checking that $f(x^*) = g(y^*)$; this is a (hopefully) quick way of demonstrating that both solutions are optimal.

Again, matchings and vertex covers are an ideal example, because the duality here is sometimes, but not always strong.

Question:	When is there strong duality between finding the largest matching in G and finding the smallest vertex cover in G?
Answer:	By Kőnig's theorem (Theorem 14.2), strong duality holds when G is bipartite.

[3] And possibly hundreds of hours more of computing time.

(By the way, you should brush up on Kőnig's theorem in preparation for the next chapter; it will come in handy!)

So how does this apply to vertex and edge connectivity? The version of the problem that exhibits duality is the problem with cuts and paths between two specific vertices s and t; the "global" version is not as nice.

Start with the problem of finding $\kappa_G(s,t)$. This is a minimization problem: we are looking for the smallest $s-t$ cut in G. The corresponding maximization problem is the problem of finding the largest set of internally disjoint $s-t$ paths. So far, we know:

Proposition 26.5. *Let s and t be two vertices in a graph G. If there is a set of k internally disjoint $s-t$ paths in G, then $\kappa_G(s,t) \geq k$: there is no $s-t$ cut with fewer than k vertices.*

Proof. Take any set U of fewer than k vertices, with $s,t \notin U$, that we suspect might be an $s-t$ cut. Every vertex $x \in U$ lies on at most one of the $s-t$ paths in our set, because they are internally disjoint. There are k paths, and fewer than k vertices in U, so there is at least one path with no vertices of U on it.

That path still exists in $G-U$, proving that s and t are in the same connected component of $G-U$. So U is not an $s-t$ cut, after all. □

Question:	What kind of duality does Proposition 26.5 give us?
Answer:	Weak duality! The proposition gives an inequality between the largest number of internally disjoint $s-t$ paths and the smallest number of vertices in an $s-t$ cut, but it does not guarantee that they meet in the middle. So far, we know of no reason why there can't be a graph G with only 10 internally disjoint $s-t$ paths, but no $s-t$ cut with fewer than 20 vertices.

Similarly, we can prove weak duality between the minimization problem of finding the smallest $s-t$ edge cut, and the maximization problem of finding the largest set of edge-disjoint $s-t$ paths:

Proposition 26.6. *Let s and t be two vertices in a graph G. If there is a set of k edge-disjoint $s-t$ paths in G, then $\kappa'_G(s,t) \geq k$: there is no $s-t$ edge cut with fewer than k edges.*

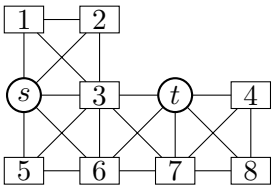
Proof. As before, but replacing vertices by edges. The key point is that every edge in G lies on at most one path in our set. A set of fewer than k edges cannot be an $s-t$ edge cut, because one of our $s-t$ paths will avoid it. □

Question:	Is there also weak duality between $s-t$ edge cuts and internally disjoint $s-t$ paths?
Answer:	Yes! Internally disjoint $s-t$ paths are, in particular, edge-disjoint.
	But whether or not strong duality will hold, we'd like to aim for the "least weak" weak duality, so we don't want to restrict our maximization problem unnecessarily. The most general sets of $s-t$ paths that will work to give a lower bound on $\kappa'_G(s,t)$ are the sets of edge-disjoint paths, so we want to use those.

In Chapter 27, we will prove that in both cases, strong duality holds: a result known as Menger's theorem.

26.5 Practice problems

1. Let G be the graph shown below.



   a) Find the vertex connectivity $\kappa(G)$.

   b) Find the $s-t$ edge connectivity $\kappa'_G(s,t)$.

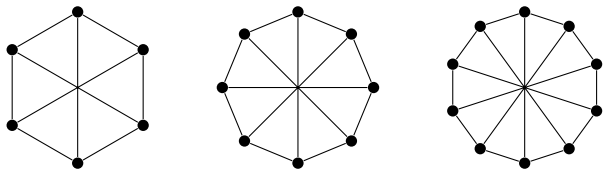
2. a) Prove that $\kappa(Q_4) = 4$ directly: by showing that no set of 3 deleted vertices can disconnect Q_4. Using Proposition 26.2 may help.

   b) Prove that $\kappa(Q_n) = n$ by induction on n, either directly or using the idea of internally disjoint paths.

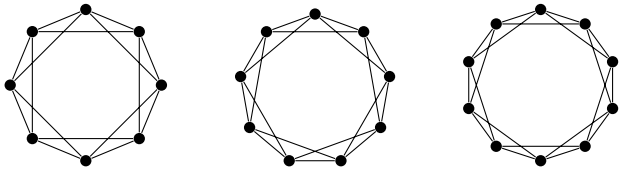
3. In Chapter 5, the *Harary graph* $H_{n,r}$ was a particular circulant graph we used in Theorem 5.1 to give an example of an r-regular graph on n vertices.

   In fact, more is true: whenever $0 \leq r \leq n-1$ and at least one of r and n is even, the Harary graph $H_{n,r}$ provides an example of an r-regular graph with $\kappa(H_{n,r}) = r$. It achieves the minimum possible number of edges in an r-connected n-vertex graph.

   a) Prove that $\kappa(H_{n,r}) = r$ for all even n when $r = 3$. The Harary graphs $H_{6,3}$, $H_{8,3}$, and $H_{10,3}$ are shown as examples below:



b) Prove that $\kappa(H_{n,r}) = r$ for all n when $r = 4$. The Harary graphs $H_{8,4}$, $H_{9,4}$, and $H_{10,4}$ are shown as examples below:



c) If you are feeling confident, prove that $\kappa(H_{n,r}) = r$ for all valid choices of n and r.

d) Give an example of a connected r-regular circulant graph which is not r-connected. (The smallest example has 8 vertices.)

4. The Petersen graph is 3-connected.

   a) Prove this by picking two vertices of the Petersen graph that are not adjacent, and finding a set of three internally disjoint paths between them.

   b) Prove this by picking a vertex of the Petersen graph, deleting it, and finding an ear decomposition of the remaining graph.

   c) Explain, using automorphisms of the Petersen graph, why either of the strategies above is a fully general proof that the Petersen graph is 3-connected.

5. A graph is called fragile if it has a vertex cut which is an independent set.

   a) Prove that the clique number of a graph is 2, then it is fragile.

   b) For all $n \geq 3$, find an example of a graph with n vertices and $2n - 3$ edges which is not fragile.

   (This exercise is based on some observations in "A note on fragile graphs" by Guantao Chen and Xingxing Yu [1].)

6. Let D be a *strongly connected digraph* with at least 3 vertices (as defined in Chapter 12), and let G be its *underlying graph* (as defined in Chapter 7). You may either assume that D never contains both arcs (x, y) and (y, x), or make G a multigraph with two copies of edge xy in this case.

   a) Prove that $\kappa'(G) \geq 2$.

   b) Give an example showing that it is not necessarily true that $\kappa(G) \geq 2$.

7. Without making use of symmetry, the strategy used to prove Proposition 26.4 would seem hopeless, because there are too many pairs (s, t) for which $\kappa(s, t)$ needs to be verified. However, it is often enough to check a smaller, but well-chosen set of pairs. The following strategy is due to Abdol Esfahanian and S. L. Hakimi [7].

   Let x be an arbitrary vertex of a graph G which is not complete, and suppose that the following is true:

   - For every vertex y not adjacent to x, $\kappa_G(x, y) \geq k$.
   - For every two vertices y, z that are adjacent to x but not to each other, $\kappa_G(y, z) \geq k$.

   Prove that $\kappa(G) \geq k$.

8. It is tempting to guess that the edge connectivity $\kappa'(G)$ is equal to $\kappa(L(G))$: the vertex connectivity of the line graph of G. However, this is false.

   a) Prove that every vertex cut in $L(G)$ corresponds to an edge cut in G. Deduce that $\kappa'(G) \leq \kappa(L(G))$.

   b) Not every edge cut in G is a vertex cut in $L(G)$. Think about how this statement might fail, and use it to find an example where $\kappa'(G) < \kappa(L(G))$. (A relatively small example exists with 5 vertices and 8 edges.)

   c) Is there a way to compute $\kappa'(G)$ using only $\kappa(L(G))$ and some simple information about G?

27 Menger's theorem

The purpose of this chapter

Menger's theorem answers the questions posed at the end of the previous chapter, and it is the main reason why $s-t$ connectivity ($\kappa_G(s,t)$ and $\kappa'_G(s,t)$ is studied): even if you want to know how hard it is to disconnect s from t for practical purposes, Menger's theorem goes a long way toward assuring you that $s-t$ connectivity is a useful model. Beyond that, it helps us understand $\kappa(G)$ (and $\kappa'(G)$) as well, often through Dirac's fan lemma.

In an introductory course on graph theory, it might be useful to have a plan for covering just one portion of this chapter, rather than the whole thing. I can think of two good ways to do this. One option is to make the proof of Menger's theorem the final goal of the semester. It is suitably dramatic, since it is a result that generalizes previous ones about matchings, answers questions posed in the previous chapter, and is one of the most difficult proofs in this book, if covered in detail.

Another option is to avoid proving Menger's theorem, and instead focus on applications of it. I've chosen to include Theorem 27.9 (the Chvátal–Erdős theorem) as an example of this, for several reasons. It connects the vertex connectivity $\kappa(G)$ to two important topics from earlier chapters: Hamilton cycles and independent sets. Also, seeing the technique in the proof (illustrated in Figure 27.4) is useful: it shows up over and over again in the study of Hamilton cycles. If taking this option, it would still be good to prove Lemma 27.2 if there's time: the connection between Menger's theorem and matchings is also an important application of Menger's theorem!

Keep in mind that this chapter relies heavily on many different concepts from previous parts of the book: Kőnig's theorem from Chapter 14, Hamilton cycles from Chapter 17, independent sets from Chapter 18, and line graphs from Chapter 20.

27.1 Menger's theorem

We ended the previous chapter with a discussion of optimization duality, and two instances of it in particular:

- The duality between minimizing the number of vertices in an $s-t$ cut, and maximizing the size of a set of internally disjoint $s-t$ paths.
- The duality between minimizing the number of edges in an $s-t$ edge cut, and maximizing the size of a set of edge-disjoint $s-t$ paths.

We proved in Proposition 26.5 and Proposition 26.6 that at the very least, *weak duality* holds in both pairs: the largest set of paths is a lower bound on the number of vertices or edges in the cut.

In 1927, Karl Menger proved [10] that *strong duality* holds in both pairs: the largest set of paths is equal to the number of vertices or edges in the cut.[4] We will begin with the vertex version of Menger's theorem. I will state it in the following form, letting Proposition 26.5 (which we've already proved) provide the converse:

Theorem 27.1 (Menger's theorem)**.** *If s and t are any two non-adjacent vertices of a graph G, then G contains a set of $\kappa_G(s,t)$ internally disjoint $s-t$ paths.*

The main goal of this chapter is to prove Menger's theorem.

Many proofs of Menger's theorem are known, so let me summarize the origin of the proof presented here. The argument by minimal counterexample in Lemmas 27.3, 27.4, and 27.5 is due to Dirac [4]. The remaining cases can be solved using Kőnig's theorem (Theorem 14.2), which is also how the proof is finished in West's *Introduction to Graph Theory* [11].

I prefer to do things this way, because Lemma 27.2 can be used to go in either direction: it can also let us deduce Kőnig's theorem from Menger's theorem. However, we do not really need the full power of Kőnig's theorem; Hall's theorem would be enough, and in fact Dirac finished the proof without using any additional results.

The relationship between Menger's theorem and our earlier results about matchings extends to algorithms. If we have a complete algorithm for computing $\kappa_G(s,t)$, which finds both an $s-t$ cut of $\kappa_G(s,t)$ vertices and a set of $\kappa_G(s,t)$ internally disjoint $s-t$ paths, then we can turn it into a matching algorithm. We do not yet have such an algorithm, and the proof of Menger's theorem in this chapter is not suitable for finding one. However, in Chapter 28, we will look at the maximum flow problem, which can be used to solve both problems. The algorithm this gives is more or less the one in Chapter 14, but it's now a special case of a more general method.

27.2 Kőnig-type graphs

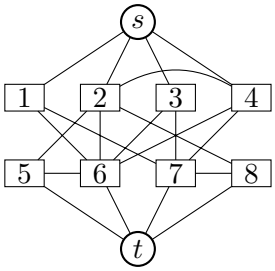
We will begin by understanding $s-t$ cuts in the special case of what I will temporarily call "Kőnig-type" graphs. This is not a standard term, just my name for the case of Menger's theorem we will address using Kőnig's theorem.

We will say that a graph G with two vertices designated s and t is *Kőnig-type* if the following is true:

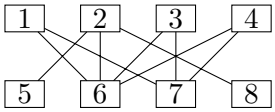
1. Vertices s and t are not adjacent. (This is a prerequisite for $\kappa_G(s,t)$ to be an interesting problem in the first place: if s and t are adjacent, then $\kappa_G(s,t)=\infty$.)
2. Every other vertex in the graph is adjacent to s or to t, but not both.

An example of a Kőnig-type graph is given in Figure 27.1a. Vertices $1,2,3,4$ are adjacent to s; vertices $5,6,7,8$ are adjacent to t.

[4] In the vertex version, we must assume that the vertices s and t are not adjacent, or else there is no $s-t$ cut at all. This will be a recurring assumption for a large part of this chapter.



(a) A Kőnig-type graph



(b) The corresponding bipartite graph

Figure 27.1: The correspondence between Kőnig-type graphs and bipartite graphs

Question:	In Figure 27.1a, the sets $\{1,2,3,4\}$ and $\{5,6,7,8\}$ are both $s-t$ cuts. Is there a smaller $s-t$ cut?
Answer:	Yes: the 3-vertex set $\{2,6,7\}$ is the unique smallest $s-t$ cut.

Question:	Is there a set of 3 internally disjoint $s-t$ paths to match it?
Answer:	Yes, represented by the walks $(s,2,5,t)$, $(s,3,6,t)$, and $(s,4,7,t)$.

Given a Kőnig-type graph G, let A be the set of vertices adjacent to s, and let B be the set of vertices adjacent to t. We construct a bipartite graph H from G by deleting vertices s and t, as well as all edges not between A and B. Figure 27.1b shows the result of applying this to the Kőnig-type graph in Figure 27.1a. (The deletion of vertices s and t is hard to miss, but edges 24, 56, and 78 have also been deleted, which is more subtle.)

Kőnig's theorem says that $\alpha'(H)$, the largest number of edges in a matching in H, is equal to $\beta(H)$, the smallest number of vertices in a vertex cover of H.

Question:	In Figure 27.1b, can you find a matching and a vertex cover of equal size?
Answer:	Edges $\{16,25,37\}$ form a matching, and vertices $\{2,6,7\}$ are a vertex cover of the same size: every edge is incident to at least one of these three vertices.

The vertex cover we found is the same set as the $s-t$ cut in Figure 27.1a. This is not a coincidence! Since $\{2,6,7\}$ is a vertex cover of H, deleting these three vertices from G also eliminates all edges between A and B, which leaves no way to get from s to t.

More precisely and more generally, let's state an equivalence:

Lemma 27.2. *Menger's theorem for a Kőnig-type graph G is equivalent to Kőnig's theorem for the associated bipartite graph H.*

Proof. We've already seen a glimpse of the relationship between $s-t$ cuts in G and vertex covers in H, but it will be easier to prove it formally if we first understand the relationship between internally disjoint $s-t$ paths in G and matchings in H.

A matching M in H, like the matching with $E(M) = \{16, 25, 37\}$ we found in Figure 27.1b, can always be turned into a set of internally disjoint $s-t$ paths of the same size. For each edge $xy \in E(M)$, take the path represented by (s, x, y, t). The internal vertices of this path are precisely the endpoints of xy, and by the definition of a matching, two edges in M share no endpoints. This means that the paths we get in this way are always internally disjoint.

Not all $s-t$ paths are like this; in Figure 27.1a, there are very long paths like the one represented by $(s, 2, 4, 7, 3, 6, 5, t)$. However, each $s-t$ path must contain an edge xy, where $x \in A$ and $y \in B$, or else it will never reach t. Given a set of internally disjoint $s-t$ paths, let M be the subgraph of H containing the first edge between A and B from each path. Because the paths were internally disjoint, these edges share no endpoints, so M is a matching.

This correspondence shows that the largest number of internally disjoint $s-t$ paths in G is equal to the matching number $\alpha'(H)$. For $s-t$ cuts in G and vertex covers in H, the relationship appears to be much more simple: they are one and the same! But why is this?

In both cases, we are looking for a set of vertices not including s or t; a subset $U \subseteq V(H)$. The definition of a vertex cover is simpler to check than the definition of an $s-t$ cut: U is a vertex cover in H if and only if it contains an endpoint of every edge xy with $x \in A$ and $y \in B$. So let's check that this also describes when U is an $s-t$ cut in G.

Question:	If U is an $s-t$ cut in G, why must it contain at least one endpoint of every edge xy with $x \in A$ and $y \in B$?
Answer:	If not, then the walk (s, x, y, t) shows that s and t are still in the same connected component of $G - U$.

Question:	If U contains at least one endpoint of every such edge, why is it an $s-t$ cut in G?
Answer:	In this case, $G - U$ has no edges between A and B (or what's left of them) remaining, so there is no way to get from s to t.

This proves that $s-t$ cuts in G are exactly the same as vertex covers in H; in particular, $\kappa'_G(s,t) = \beta(H)$.

We're now ready to prove the equivalence claimed in the lemma. We have two pairs of equal quantities, one in G and one in H:

1. The largest size of a set of internally disjoint $s-t$ paths in G is equal to the matching number $\alpha'(H)$.
2. The $s-t$ connectivity $\kappa_G(s,t)$ is equal to the vertex cover number $\beta(H)$.

Both Menger's theorem for G and Kőnig's theorem for H say that the first pair is also equal to the second pair, so the claims made by the two theorems are equivalent. □

27.3 Reducing all other cases

To obtain a proof of Menger's theorem in general, we must now deal with graphs which are not Kőnig-type.

We will use a proof by minimal counterexample: a sort of combination of a proof by induction and a proof by contradiction. Suppose for sake of contradiction that Menger's theorem is false for some graphs. Then we may take G to be a counterexample to Menger's theorem which has as few vertices as possible: a minimal counterexample.

Sometimes a minimal counterexample is also called a "minimal criminal", but I don't like this terminology, because it's too much of a tongue-twister when I try to say it out loud while teaching class. I don't mind writing it in a book, though.

What do we do with a minimal counterexample? We try to deduce additional properties: for example, right now, in the case of Menger's theorem, we know that a minimal counterexample is not Kőnig-type, because then it wouldn't be a counterexample. The reason minimality is so important is that very often, we can say, "A minimal counterexample must have property X, because otherwise, we could modify it in Y way and obtain a smaller counterexample."

Proofs by minimal counterexample can often be turned into proofs by induction, but I want to include a proof of this type here, because they're a very important exploration technique. When you don't yet know whether a theorem is true, proving some properties of a minimal counterexample is useful in two ways: it is partial progress toward a proof of the theorem (if it's true), but it can also help us find a counterexample (if it's false).

So let's see how it works here. The following three lemmas all have the common hypothesis that a graph G, with designated non-adjacent vertices s and t, is a minimal counterexample to Menger's theorem. (I will not keep restating these assumptions about G each time.) We will prove a few properties of G, and then put them together to prove Menger's theorem. I should warn you that (as in every book in which the hero has to accomplish three tasks) the third lemma is the trickiest.

Lemma 27.3. *G cannot contain a vertex x adjacent to both s and t.*

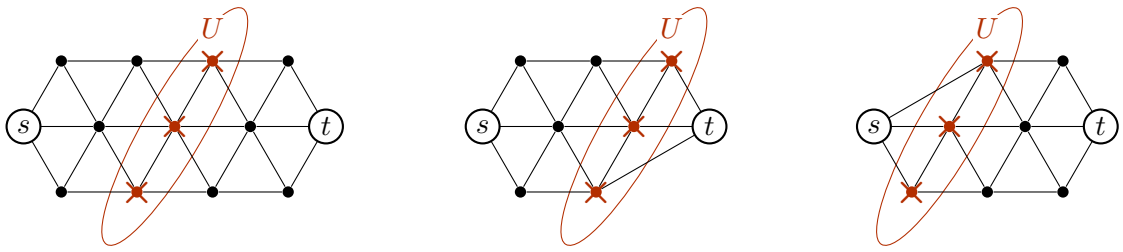
Proof. Suppose for the sake of contradiction that G has such a vertex. Then x must be part of every $s-t$ cut: if it is not deleted, there is an $s-t$ path P represented by (s,x,t). So every $s-t$ cut has the form $U \cup \{x\}$, where U is an $s-t$ cut in $G-x$.

Let $H = G - x$. Because (as we've just proved) every $s-t$ cut in G consists of an $s-t$ cut in H, plus one extra vertex, we have $\kappa_G(s,t) = \kappa_H(s,t) + 1$.

But we know that H is smaller than the minimal counterexample to Menger's theorem! So by Menger's theorem, H contains a set of $\kappa_H(s,t)$ internally disjoint $s-t$ paths.

All these $s-t$ paths are also $s-t$ paths in G, and we can add one more path to the set: the path P defined above. Therefore G contains a set of $\kappa_H(s,t)+1$ internally disjoint $s-t$ paths. But this is exactly the number of paths Menger's theorem promises, so we contradict our assumption that G is a counterexample to Menger's theorem. □

Lemma 27.4. *Every vertex in G, other than s and t, is part of some $s-t$ cut of size $\kappa_G(s,t)$.*



(a) The original graph G (b) Subproblem 1 (L) (c) Subproblem 2 (R)

Figure 27.2: The subproblems in Lemma 27.5

Proof. Let x be an arbitrary vertex of G, and let $H = G - x$. As in the previous lemma, H is smaller than the minimal counterexample to Menger's theorem, so H contains a set of $\kappa_H(s,t)$ internally disjoint $s-t$ paths.

Therefore G also contains a set of $\kappa_H(s,t)$ internally disjoint $s-t$ paths. Since G is a counterexample to Menger's theorem, these paths must not be enough: $\kappa_G(s,t)$ must be bigger than $\kappa_H(s,t)$. In other words, $\kappa_G(s,t) \geq \kappa_H(s,t) + 1$.

Let U be an $s-t$ cut in H of size $\kappa_H(s,t)$. Then $U \cup \{x\}$ is an $s-t$ cut in G of size $\kappa_H(s,t)+1$, because $G-(U\cup\{x\})$ is exactly the same graph as $H-U$. This proves that $\kappa_G(s,t) \leq \kappa_H(s,t)+1$, and since we have the reverse inequality already, we know that $\kappa_G(s,t)$ and $\kappa_H(s,t)+1$ are equal.

Therefore $U\cup\{x\}$ is the $s-t$ cut containing x we wanted: it has size $\kappa_H(s,t)+1 = \kappa_G(s,t)$. □

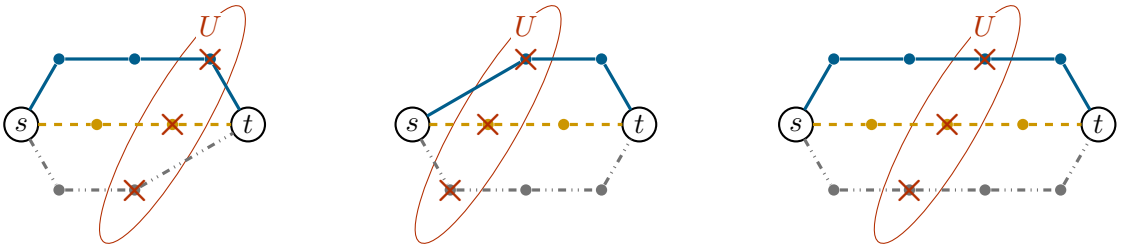
Let A be the set of all vertices adjacent to s in G, and let B be the set of all vertices adjacent to t. From Lemma 27.3, we know that A and B are disjoint, but we do not (yet) know if they include every vertex other than s and t. Both A and B are $s-t$ cuts, though they might have more than $\kappa_G(s,t)$ vertices.

Lemma 27.5. *G has no $s-t$ cuts of size $\kappa_G(s,t)$ not equal to either A or B.*

Proof. Let U be an $s-t$ cut in G with $|U| = \kappa_G(s,t)$. If U contains all of A, then U must be equal to A, because otherwise A would be a smaller $s-t$ cut. Similarly, if U contains all of B, then U must be equal to B. So assume, for the sake of contradiction, that U does not contain all of either set: there is some $x \in A$ and some $y \in B$ such that U does not contain either x or y.

The vertices x and y will be important eventually, but for the moment, we simply use U to split up the problem of finding internally disjoint $s-t$ paths in G into two smaller subproblems. This is shown by example in Figure 27.2, which is worth a thousand words. Here are a few words to describe what we do in general:

Subproblem 1. Take the subgraph of G induced by U and all the vertices in the same connected component of $G-U$ as s. This does not include t, because in $G-U$, vertices s and t are in different components. But now, add t back in, with edges from every vertex in U directly to t. Call the resulting graph L (for "left", since in our example, it retains the left half of Figure 27.3).



(a) $s-t$ paths in L (b) $s-t$ paths in R (c) $s-t$ paths in G

Figure 27.3: Recombining the subproblems in Lemma 27.5

Subproblem 2. Take the subgraph of G induced by U and all the vertices in a different connected component of $G-U$ from s. (This includes t, but not s.) Add s back in, with edges from s directly to every vertex in U. Call the resulting graph R (for "right").

The vertices x (in A, but not in U) and y (in B, but not in U) guarantee that the two subproblems are strictly smaller than G: in L, there is no vertex y, and in R, there is no vertex x. As in Lemma 27.3, the natural next step is to apply Menger's theorem to L and R. To do this, the first thing we need to know is $\kappa_L(s,t)$ and $\kappa_R(s,t)$.

Question:	In Figure 27.2, how do $\kappa_L(s,t)$ and $\kappa_R(s,t)$ compare to $\kappa_G(s,t)$?
Answer:	All three $s-t$ connectivities are equal.

This is true in general. To find an $s-t$ cut in L, we must block all ways to get from s to U (since every vertex to U is adjacent to t). This also blocks all ways to get from s to U in G (since before we pass through U, we cannot reach a place where G and L disagree), so we get an $s-t$ cut in G as well, proving $\kappa_L(s,t) \geq \kappa_G(s,t)$. Similarly, $\kappa_R(s,t) \geq \kappa_G(s,t)$, because an $s-t$ cut in R blocks all ways to get from U to t, which makes it an $s-t$ cut in G as well.

The reverse inequalities $\kappa_L(s,t) \leq \kappa_G(s,t)$ and $\kappa_R(s,t) \leq \kappa_G(s,t)$ hold because U is a cut of size $\kappa_G(s,t)$ in all three graphs: G, L, and R.

Because L is strictly smaller than G, Menger's theorem holds for L, giving a set of $\kappa_G(s,t)$ internally disjoint $s-t$ paths in L. Since $|U| = \kappa_G(s,t)$ and every $s-t$ path in L must first pass through U, this set of paths uses each vertex in U exactly once. By removing the last step from each path, we get a set of internally disjoint paths from s to every vertex of U.

Because R is strictly smaller than G, Menger's theorem holds for R, giving a set of $\kappa_G(s,t)$ internally disjoint $s-t$ paths in R. Since $|U| = \kappa_G(s,t)$ and every $s-t$ path in R must first pass through U, this set of paths uses each vertex in U exactly once. By removing the first step from each path, we get a set of internally disjoint paths from every vertex of U to t.

For each vertex $u \in U$, the union of an $s-u$ path from the first set and a $u-t$ path from the second set is an $s-t$ path in G. (This is shown for our example in Figure 27.3.) All $\kappa_G(s,t)$ paths obtained in this way are internally disjoint: they cannot intersect in or before U, because that would be an intersection in L, and they cannot intersect after U, because that would be an intersection in R.

But finding a set of $\kappa_G(s,t)$ internally disjoint $s-t$ paths in G contradicts our choice of G: that it was a minimal counterexample to Menger's theorem. Therefore the $s-t$ cut U, which we assumed to exist at the beginning of this proof, cannot exist in G. □

Take a deep breath: Lemma 27.3 was quick to prove, and Lemma 27.4 was not bad either, but Lemma 27.5 might have been more than we bargained for. But now, we are ready to finish the proof of Menger's theorem in just a few more steps!

Proof of Menger's theorem (Theorem 27.1). Suppose for the sake of contradiction that a minimal counterexample to Menger's theorem exists: a graph G with designated not-adjacent vertices s and t which does not have a set of $\kappa_G(s,t)$ internally disjoint $s-t$ paths.

By Lemma 27.2 and Kőnig's theorem, we know that G (a counterexample to Menger's theorem) cannot be a Kőnig-type graph.

Question:	Looking back at the definition of Kőnig-type graphs earlier in this chapter, what does it mean that G is not one?
Answer:	The condition for G to be Kőnig-type is that every vertex other than s or t is adjacent to s or t, but not both. So if G is not Kőnig-type, it must have a vertex (other than s or t) which is either adjacent to both s and t, or to neither.

Let x be one such vertex. If x is adjacent to both s and t, this contradicts Lemma 27.3. If x is adjacent to neither s nor t, then we need to work harder. First, apply Lemma 27.4 to find a vertex cut U of size $\kappa_G(s,t)$ with $x \in U$.

Question:	Why does this contradict Lemma 27.5?
Answer:	Lemma 27.5 says that in this case, $U = A$ (the set of neighbors of s) or $U = B$ (the set of neighbors of t). But neither one is possible, because $x \in U$ and x is not adjacent to s or t.

In all case, G contradicts one of the lemmas we proved, so we conclude that a counterexample to Menger's theorem does not exist. □

27.4 Extensions

Many variants of Menger's theorem exist. To begin with, a version of the theorem for $s-t$ edge cuts also exists, which is the converse of Proposition 26.6.

Theorem 27.6 (Menger's theorem, edge version)**.** *If s and t are any two vertices of a graph G, then G contains a set of $\kappa'_G(s,t)$ edge-disjoint $s-t$ paths.*

It is very tempting (and almost works) to try to prove this version of Menger's theorem by applying Theorem 27.1 to the line graph $L(G)$.

Question:	What is the main problem with trying to relate $\kappa'_G(s,t)$ to $\kappa_{L(G)}(s,t)$?
Answer:	The line graph $L(G)$ doesn't have vertices called s and t: its vertices are edges of the original graph!

To fix the problem, we must modify $L(G)$ a little.

Proof of Theorem 27.6. Define a new graph H in one of the following two equivalent ways:

- Construct G' from G by adding two new vertices: one adjacent only to s and one adjacent only to t. Then, let $H = L(G)$.
- Construct H from $L(G)$ by adding two new vertices s^* to t^*. For every vertex of $L(G)$ which represents an edge incident to s, make it adjacent to s^* in H; for every vertex of $L(G)$ which represents an edge incident to T, make it adjacent to t^* in H.

Question:	If we construct H in the first way, what are the two vertices of H that correspond to s^* and t^*?
Answer:	They come from the edges of G' incident to the two newly-added vertices.

We must resolve an issue that will otherwise repeatedly bother us throughout this proof: the relationship between $s^* - t^*$ paths in H and $s - t$ paths in G. In one direction, things work out nicely. Given a walk $(x_0, x_1, \ldots, x_l)$ in G where $x_0 = s$ and $x_1 = t$, the sequence $(s^*, x_0x_1, x_1x_2, \ldots, x_{\ell-1}x_\ell, t^*)$ is an $s^* - t^*$ walk in H, and if the first walk represents a path, so does the second. (Check this!)

Question:	What about the other direction; is it true that every $s^* - t^*$ path in H can be turned into an $s - t$ path in G by reversing this process?
Answer:	No! If it's been a while since you thought about paths in $L(G)$, you should go back and re-read Chapter 20, but walks in a line graph can do some things that walks in the original graph cannot: they can contain segments of the form $(\ldots, xy, xz, xw, \ldots)$ with several edges all incident to the same vertex of G.

However, given an $s^* - t^*$ path in H, we can look at the subgraph of G containing all the edges that were internal vertices of the $s^* - t^*$ path, and all their endpoints. This is a connected subgraph of G containing s and t, so within it we can find an $s - t$ path.

By applying Menger's theorem to H, we get a set of $\kappa_H(s^*, t^*)$ internally disjoint $s^* - t^*$ paths. We know how to turn each of them into an $s - t$ path in G, but we want to know that we'll get a set of edge-disjoint paths when we do so.

Question:	Suppose P and Q are two internally disjoint $s^* - t^*$ paths in H. When we use them to find two $s - t$ paths in G, why are those paths edge-disjoint?
Answer:	Our first steps with P and Q are to go to connected subgraphs of G whose edges correspond to internal vertices of P and of Q. So even those connected subgraphs have no edges in common! Therefore the paths we find in them are also edge-disjoint.

We've found a set of $\kappa_H(s^*, t^*)$ edge-disjoint $s - t$ paths in G, but that's not enough. We need $\kappa'_G(s, t)$ edge-disjoint $s - t$ paths, so we must show that we have at least as many as we need: that $\kappa_H(s^*, t^*) \geq \kappa'_G(s, t)$. (The two should be equal, but proving that is more than we need.)

In other words, we need to show that every $s^* - t^*$ cut U in H is an $s - t$ edge cut in G. By definition of an $s^* - t^*$ cut, every $s^* - t^*$ path in H passes through some element of U. We've already seen that every $s - t$ path in G corresponds to an $s^* - t^*$ path in H: so every $s - t$ path in G uses an edge of U. That's exactly what it means for U to be an $s - t$ edge cut!

Therefore applying Menger's theorem to H and turning the paths we get back into paths in G, we get a set of at least $\kappa'_G(s, t)$ edge-disjoint $s - t$ paths in G, proving the theorem. □

Both versions of Menger's theorem work for multigraphs and for directed graphs. In the case of multigraphs, Theorem 27.1 is not at all interesting: loops and parallel edges don't help us get more internally disjoint paths, and don't affect vertex cuts. However, Theorem 27.6 can be usefully applied to multigraphs: for example, if we have 17 parallel edges incident to x and y, that lets us have 17 edge-disjoint paths all taking a step from x to y. The proof we gave of Theorem 27.6 works just fine in this case.

Question:	In the case of directed graphs, the notion of connected components is much less clear, so how should we define $s - t$ cuts?
Answer:	Removing an $s - t$ cut (either a vertex cut, or an "arc cut") from a directed graph should destroy all directed paths from s to t.

Using Menger's theorem for directed graphs can be useful for both vertex and edge cuts, but it takes some work to obtain. In principle, the proof strategy we used for Menger's theorem still works for directed graphs. However, some statements need to be more precise: for example, a vertex x should be considered "adjacent to s" if there is an arc from s to x, but it should be considered "adjacent to t" if there is an arc from x to t. The sets A and B are defined accordingly.

Question:	How should we modify the construction of the associated bipartite graph H of a Kőnig-type graph G when G is a directed graph?
Answer:	The graph H should remain undirected; however, it should only have an edge xy with $x \in A$ and $y \in B$ if the directed graph G has an arc from x to y.

That's it for the various versions of Menger's theorem, but there's a bit more to be said about its extensions.

First of all, you may feel a bit uncomfortable about the way we've completely forgotten about the global parameters $\kappa(G)$ and $\kappa'(G)$, choosing to focus on $\kappa_G(s,t)$ and $\kappa'_G(s,t)$. Well, we can now return to the global parameters, because for all vertices $s,t \in V(G)$, we have $\kappa_G(s,t) \geq \kappa(G)$ and $\kappa'_G(s,t) \geq \kappa'(G)$. For instance, we can immediately get the following corollary:

Corollary 27.7. *If G is a k-connected graph and s and t are any two non-adjacent vertices in G, then G contains a set of k internally disjoint $s-t$ paths.*

Given a graph G, a vertex $s \in V(G)$, and a subset $T \subseteq V(G)$ with $s \notin T$, we define an *$s-T$ fan* in G to be a set of paths from s to vertices in T that share no vertices except s. (In particular, their ends in T must be distinct.) To be clear, the size of an $s-T$ fan is the number of paths in the fan.

The notion of $s-T$ fans was introduced by Dirac [5]. I suspect that the idea later helped Dirac arrive at the proof of Menger's theorem in this chapter, because Lemma 27.5 is all about combining an $s-U$ fan and a $t-U$ fan to get a set of internally disjoint $s-t$ paths. But even before that, Dirac proved the following lemma as a consequence of Menger's theorem.

Lemma 27.8 (Dirac's fan lemma)**.** *If G is a k-connected graph, $T \subseteq V(G)$, and s is a vertex not in T, then G contains an $s-T$ fan of size* $\min\{|T|, k\}$.

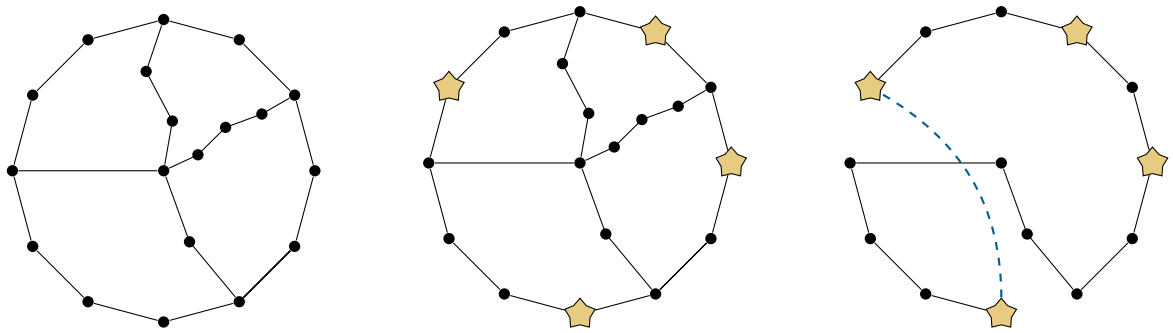
Proof. First, let's deal with the case $|T| \geq k$, in which case we want to find an $s-T$ fan of size k. Define a new graph H from G by adding a new vertex t adjacent to every element of T.

First, we prove that $\kappa_H(s,t) \geq k$. Suppose not: suppose that U is an $s-t$ cut in H with $|U| \leq k-1$. Then $|U| < |T|$, so in $H-U$, there must be at least some vertices of T left, which are in the same connected component as T, and therefore in a different connected component from s. This means that in $G-U$, those vertices of T are still in a different component from s, which makes U a vertex cut in G. But this contradicts our assumption that G is k-connected.

By applying Menger's theorem to H, we obtain a set of at least k internally disjoint $s-t$ paths in H. By removing vertex t from k of these paths, what we get is exactly an $s-T$ fan in G of size k.

If $|T| < k$, then we first replace T by a set T' containing T of size exactly k. (A k-connected graph must have at least $k+1$ vertices, so this is always possible.) If we find an $s-T'$ fan in G of size k, then it contains a path from s to every vertex of T'; in particular, to every vertex of T. By taking only the paths that go to T, we get an $s-T$ fan of size $|T|$. □

By the way, Dirac's fan lemma is so commonly used that among graph theorists familiar with it, it is often just called the fan lemma. But I wanted to disambiguate because I've seen this confuse mathematicians outside this specific subfield of graph theory: there is also a result in combinatorial topology called the Ky Fan lemma. (Here, Fan is a last name.)



(a) A cycle and a fan (b) Vertices x_{i+1}, where $i \in I$ (c) A longer cycle is found

Figure 27.4: Extending the cycle in Theorem 27.9

27.5 Cuts and long cycles

Let's end this chapter by seeing an application of connectivity and of Dirac's fan lemma to some topics covered earlier in this book. The following theorem was proved in 1972 by Václav Chvátal and Pál Erdős [2], both of whom we've met before:

Theorem 27.9. *If G is a graph with at least 3 vertices such that the connectivity $\kappa(G)$ is at least the independence number $\alpha(G)$, then G is Hamiltonian.*

Proof. If $\alpha(G) = 1$, then G can't be missing even a single edge between its vertices, in which case it's definitely Hamiltonian; so we will assume $\alpha(G) \geq 2$ and $\kappa(G) \geq 2$. It's a good sign that G is 2-connected; this means that at least it has some cycles, by any of the results in Chapter 25. This proof will proceed by finding longer and longer cycles, until finally getting a Hamilton cycle.

Let C be the longest cycle we've found so far. If we're not done with the proof yet, we assume that it is not a Hamilton cycle: there is some vertex $y \notin V(C)$. Our goal will be to find a longer cycle in G, and to make sure it's longer, we will find a cycle that passes through every vertex in $V(C)$ and also passes through y.

Take a $y - V(C)$ fan in G with as many paths in it as possible; by Dirac's fan lemma, it has size at least $\min\{|C|, \kappa(G)\}$, but if we're lucky, it might be larger. An example is shown in Figure 27.4a (with y in the center). The definition of a fan does not guarantee that the paths it contains do not share any internal vertices with C, though I've drawn the fan in this way. However, we can assume that this is true anyway, by stopping every path in the fan as soon as it reaches a vertex of C.

Let $(x_0, x_1, \ldots, x_{l-1}, x_0)$ be a walk representing C. Choosing this walk representation also gives us a direction to follow along C. If all we have is the cycle, all we can say is that every vertex on the cycle has two neighbors. With the walk, we can say that every vertex $x_i \in V(C)$ has a vertex x_{i+1} that comes after it (taking $x_l = x_0$) and a vertex x_{i-1} that comes before it (taking $x_{-1} = x_{l-1}$).

This is important, because I want to define a set of vertices in an usual way. First, let $I \subseteq \{0, 1, \ldots, l-1\}$ be the set of positions in the cycle where some path in the $y - V(C)$ fan ends: that is, the fan contains a $y - x_i$ path whenever $i \in I$, and not otherwise. Now, instead of

looking at the vertices in the fan, look at the vertices immediately following them on the path: the set $\{x_{i+1} : i \in I\}$. These are marked in Figure 27.4b.

Looking at this set of vertices appears unmotivated at first, but it's a common trick in results about Hamilton cycles, so I want to explain it a bit. We will soon be trying to find a cycle that mostly consists of the edges shown in Figure 27.4a; although G has many other edges not shown in the diagram, we don't know much about those edges. In this figure, each vertex x_i with $i \in I$ has degree 3, so a cycle through them will use two of their neighbors and neglect the third. Sometimes, x_{i-1} or x_{i+1} might be that neglected neighbor, and so we look at it more closely to find some other edge it might have!

In fact, to get a cycle through y and every vertex of C, it's enough to find a single edge $x_{i+1}x_{j+1}$ where $i, j \in I$. An example of such a cycle is shown in Figure 27.4c. More formally, the cycle is obtained as follows:

1. By deleting edges x_ix_{i+1} and x_jx_{j+1} from C, we break it into two pieces: an $x_{i+1} - x_j$ path P and an $x_i - x_{j+1}$ path Q. Together, $V(P) \cup V(Q) = V(C)$.
2. By combining the $y - x_i$ and $y - x_j$ paths in the fan, we get an $x_i - x_j$ path R that passes through y and shares none of its internal vertices with P or Q.
3. The union $P \cup Q \cup R$ combines an $x_{i+1} - x_j$ path, an $x_j - x_i$ path, and an $x_i - x_{j+1}$ path; these paths share no vertices apart from x_i and x_j. Therefore $P \cup Q \cup R$ is an $x_{i+1} - x_{j+1}$ path. Adding the edge $x_{j+1}x_{i+1}$ gives us the cycle we wanted!

Question:	One special case of this might be confusing: the case where $j = i + 1$, so that the edge $x_{i+1}x_{j+1}$ is actually the edge $x_{i+1}x_{j+2}$ which is part of the cycle we started with. What do we do in this case?
Answer:	In this case, the path R we defined is an $x_i - x_{i+1}$ path passing through y which is internally disjoint from C, so it can be inserted into the middle of the cycle between x_i and x_{i+1} to get a longer cycle. The same works if $i = l - 1$ and $j = 0$.

We've almost finished the proof, except for one detail: how do we guarantee that an edge $x_{i+1}x_{j+1}$ exists? I have to wonder if Chvátal and Erdős came up with the proof up until now, and only then asked themselves this question, and that's how they came up with the hypotheses of this theorem. After all, we've barely used the assumption about the independence number of G, so far!

There are two cases. If $|V(C)| \leq \kappa(G)$, then Dirac's fan lemma guarantees a fan of size $|V(C)|$, which includes a path to every vertex of C. Therefore every edge of the cycle is an edge of the type we're looking for! For example, the edge x_1x_2 will work, for $i = 0$ and $j = 1$, because our fan includes an $y - x_0$ path and a $y - x_1$ path.

If $|C| > \kappa(G)$, then Dirac's fan lemma only guarantees a fan of size $\kappa(G)$. At this point, we remember that $\kappa(G) \geq \alpha(G)$. If only we had $\kappa(G) > \alpha(G)$, instead! Then we'd know that the set $\{x_{i+1} : i \in I\}$ cannot be independent, because it has size $\kappa(G)$: it's bigger than the largest independent set.

But a theorem with $\kappa(G) > \alpha(G)$ as a hypothesis would not be the best theorem Chvátal and Erdős could prove. We can still finish the proof with the hypothesis $\kappa(G) \geq \alpha(G)$. For this, consider the set $\{x_{i+1} : i \in I\} \cup \{y\}$, which has at least $\kappa(G) + 1$ vertices, and therefore isn't independent. We either get an edge $x_{i+1}x_{j+1}$ with $i, j \in I$, which is what we wanted, or an edge $x_{i+1}y$ with $i \in I$, which... isn't what we wanted.

Question:	Can you think of what to do with the edge $x_{i+1}y$?
Answer:	By combining it with the $y - x_i$ path in the fan, we get an $x_i - x_{i+1}$ path through y, internally disjoint to C. This can be inserted into the cycle in place of going from x_i to x_{i+1}, obtaining a longer cycle.

To sum it up: we started with an arbitrary cycle which was not a Hamilton cycle, and were able to make it longer, by incorporating at least one additional vertex. We can repeat this to grow the new cycle, too, as many times as necessary. The only way this process can end is by finding us a Hamilton cycle, proving that G is Hamiltonian. □

You might wonder if the difference between the hypothesis "$\kappa(G) \geq \alpha(G)$" in Theorem 27.9, and the hypothesis "$\kappa(G) > \alpha(G)$" with which we could have ended the proof early, is really all that big. Of course, a theorem with the hypothesis $\kappa(G) \geq \alpha(G)$ is better, because it applies to more graphs, but is it a lot better?

One point in favor of the slightly-better result is that it's the best result possible: the theorem could not be improved even further, to work with the hypothesis $\kappa(G) \geq \alpha(G)-1$. One example showing that this is impossible is the Petersen graph: it is 3-connected (by a practice problem in Chapter 26) and has independence number 4 (in the usual diagram with two 5-cycles, an independent set can contain at most two vertices from each cycle). However, the Petersen graph is not Hamiltonian (by Proposition 17.1). So the hypothesis $\kappa(G) \geq \alpha(G) - 1$ is not always strong enough to guarantee a Hamilton cycle!

27.6 Practice problems

1. There are 16 computers in a network with the IDs

   $$10, 12, 13, 14, 20, 21, 23, 24, 30, 31, 32, 34, 40, 41, 42, 43$$

   consisting of 2-digit numbers with digits 0 through 4, not both the same. Two computers whose IDs have the same first digit are directly connected. Additionally, two computers whose IDs have the same two digits in a different order (such as 24 and 42) have two direct connections between them, one serving as a backup in case the other fails.

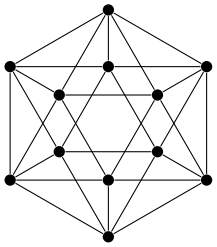
   Computers with no direct connection can still communicate by relaying messages through a sequence of directly connected computers.

   a) You are at computer 12 and your friend is at computer 34. How many intermediate computers, at minimum, need to fail (and stop relaying messages) before you can no longer communicate with your friend?

b) How many connections between computers would need to fail before you can no longer communicate with your friend?

Justify both answers by an example, as well as a set of paths demonstrating that the example is the minimum possible.

2. Prove that the skeleton graph of the icosahedron (shown below) is 5-connected by finding a set of 5 internally disjoint $s - t$ paths in all three cases: (a) when s and t are adjacent, (b) when the distance $d(s, t)$ is 2, and (c) when the distance $d(s, t)$ is 3.



Why is the graph not 6-connected?

3. Prove that the hypothesis of Theorem 27.9 is the best possible hypothesis for all values of $\kappa(G)$. That is, for all $k \geq 1$, find a graph G with $\kappa(G) = k$ and $\alpha(G) = k + 1$ which is not Hamiltonian.

4. Prove Menger's theorem using Lemma 27.3, Lemma 27.4, and Lemma 27.5, and Hall's theorem (Theorem 15.1), but without using Kőnig's theorem.

5. Many of the results about 2-connected graphs in Chapter 25 can now be obtained much more easily using the results in this chapter.

   a) Prove Theorem 25.1 that any two vertices in a 2-connected graph lie on a common cycle, by using Corollary 27.7.

   b) Prove Lemma 25.7 that for any three vertices u, v, x in a 2-connected graph, there is a $u - v$ path that passes through x, by using Dirac's fan lemma.

6. Prove that if G is 3-connected, then for any three vertices, G contains a cycle that passes through all of them.

   More generally, it is true that if G is k-connected, then for any k vertices, G contains a cycle through all k of them (in some order); this was the theorem that Dirac invented his fan lemma to prove. If you're brave, try proving this by induction on k.

28 Maximum flows

The purpose of this chapter

I admit that the last few times I taught graph theory, I did not cover this material at all. It's not that it's not important. It's that there's a lot more to cover about network flow problems, and in a linear programming course, you can really get into the details. (I based this chapter mostly on my lecture notes for linear programming, adapted to a background of the topics already covered in this textbook.)

However, this topic is a nice hands-on counterpart to the highly theoretical Chapter 27. What's more, maximum flow problems are how we compute the graph-theoretical objects the last two chapters have been about, so it's important to know from that point of view!

The proof of Theorem 28.7 is something I added to this chapter after looking it up in the original paper of Edmonds and Karp, but it's much nicer than I expected it to be; if I were teaching a class on this material again, I would definitely see if I could include it.

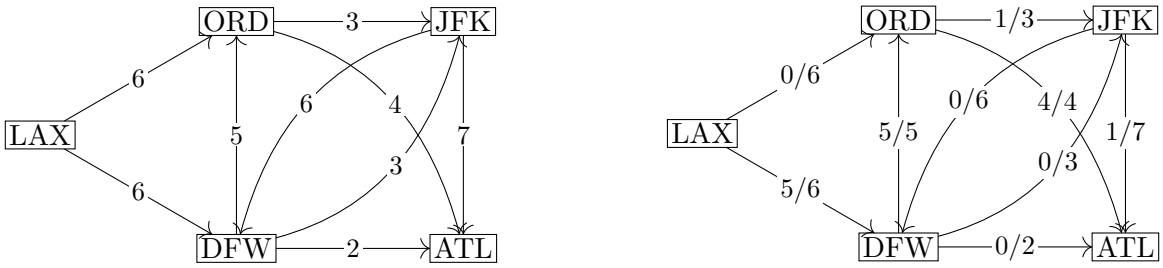
Networks are directed graphs; even though I will not need to use any theorems specific to directed graphs, you should make sure you're at least comfortable with the directed graph terminology in Chapter 7. Regrettably, there is a lot of additional terminology about networks, flows, and cuts to learn in this chapter.

28.1 Shipping oranges

Suppose you grow oranges in California and want to ship them by airplane to Atlanta. Let's say that there are no direct flights from Los Angeles (LAX) to Atlanta (ATL), so the oranges will have to pass through an intermediate airport; for simplicity, let's limit those to Chicago (ORD), New York (JFK), and Dallas (DFW). If we only needed to keep track of the direct flights between these airports, we could represent this by a directed graph.

Such a directed graph is shown in Figure 28.1a, but with additional information: the arcs are labeled with numbers. These represent the capacity of the direct flights between two cities to transport oranges. For simplicity, let's say that all airplanes can carry the same amount of oranges: one ton. Then the capacity can be measured in tons (per day) or flights (per day) equally well, and this is what the numbers in Figure 28.1a mean. We could also replace an arc with capacity 6 by six arcs with capacity 1, and thereby eliminate capacities entirely, but this would make the diagram much busier—and won't work well if, in the future, we end up with fractional capacities.

How can we transport as many oranges as we can from LAX to ATL? Figure 28.1b shows one feasible (but not particularly good) solution, which transports 5 tons of oranges from LAX



(a) A network with capacities

(b) A LAX−ATL flow

Figure 28.1: Shipping oranges to Atlanta

to ATL per day. In words: we fly all 5 tons of oranges to DFW, and then fly all 5 tons to ORD. From there, we split them up: 4 tons of oranges fly directly to ATL, and 1 ton is routed to ATL via JFK.

In the diagram, each arc is marked with two numbers, and you should think of "5/6" not as the fraction $\frac{5}{6}$ but as "5 out of 6". Each arc's label is $f(x, y)/c(x, y)$, where $f(x, y)$ represents the number of flights along that route which we actually use to ship oranges.

What are the constraints on the values $f(x, y)$? The most straightforward constraints are the ones coming from the numbers in the diagram: for example, we should have $f(\text{LAX}, \text{DFW})$ should be between 0 and 6 because there are only 6 flights from LAX to DFW each day, and we can use between 0 and 6 of them to ship oranges. But if those were all the constraints, then we'd "solve" the problem by setting each variable to its maximum value—after all, why not?

Question:	What stops us from doing that?
Answer:	Conservation of mass; more precisely, conservation of oranges. For example, the total capacity of the arcs leaving LAX is 12, but the total capacity of the arcs entering ATL is 13. If we were to set all variables to their maximum value, we'd be shipping 12 tons of oranges out of LAX and shipping 13 tons of oranges into ATL, each day. Where does the extra ton of oranges come from?

We need to add constraints saying that oranges can't appear out of nowhere or vanish into nowhere. The "conservation of oranges" constraints will look at every intermediate airport and say: the number of oranges going in should equal the number of oranges going out. For example, at JFK, this constraint would be

$$f(\text{ORD}, \text{JFK}) + f(\text{DFW}, \text{JFK}) = f(\text{JFK}, \text{DFW}) + f(\text{JFK}, \text{ATL}).$$

We do not have such a constraint at LAX or ATL. There are no oranges that can be shipped into LAX, but that does not mean that $f(\text{LAX}, \text{ORD}) + f(\text{LAX}, \text{DFW})$ (the number of oranges shipped out of LAX) should be 0: in fact, we want this quantity to be as large as possible! To solve our problem, we can either maximize $f(\text{LAX}, \text{ORD}) + f(\text{LAX}, \text{DFW})$ or, equivalently, maximize $f(\text{ORD}, \text{ATL}) + f(\text{JFK}, \text{ATL}) + f(\text{DFW}, \text{ATL})$: the number of oranges arriving in ATL. With the "conservation of oranges" constraint in play, these should be one and the same.

Now we have all the ingredients we need for this optimization problem, which is our first example of a maximum flow problem.

Question:	Just for fun: what is the maximum amount of oranges we can ship per day in the problem as shown in Figure 28.1?
Answer:	The maximum is 12 tons per day. Ship 6 tons from LAX to ORD, and then split them up, with 4 going to ATL and the other 2 going to JFK. Also, ship 6 tons from LAX to DFW, and then split them up, with 2 going to ATL and the other 4 going to JFK. This gets 6 tons to ATL, and 6 more to JFK, which can then be sent to ATL as well.

28.2 Maximum flow problems

Now, let's describe everything we've done in more general terms rather than in oranges.

First, we will define what a network is. Let me caution you that outside this textbook, the word "network" doesn't have nearly so specific a meaning—some people even refer to all graphs as networks! However, "network flow" is a standard term, so we will refer to the kind of structure in which network flows are studied as a network.

A *network* is a directed graph N with two additional pieces of information:

- A non-negative value $c(x, y)$ associated to each arc (x, y). Rather than refer to $c(x, y)$ as a "weight" or "cost" as we might usually, we say that it is the *capacity* of arc (x, y).
- Two special vertices: a *source* s and a *sink* t. We will assume that these are also a source and a sink in the sense we used these words in Chapter 7: s has indegree 0 and t has outdegree 0.

Question:	If we are trying to ship oranges (for example) from s to t, why does it make sense for s to have indegree 0 and for t to have outdegree 0?
Answer:	We will never need to ship along arcs into s, because that would mean the oranges made a full circle and returned where we started, so we might as well ignore such arcs—and the same goes for arcs out of t.

Question:	Can we assume that s and t are the only vertices with indegree or outdegree 0?
Answer:	Yes. If there were another vertex x with indegree 0, we could never get oranges to x. If there were another vertex y with outdegree 0, even if we did get oranges to y, we wouldn't want to, because we couldn't get them out again. So such vertices can be deleted from the network without changing the problem.

Now we know what a network is. A *flow* in a network N is an assignment of a non-negative real value $f(x, y)$ to every arc $(x, y) \in E(N)$; we refer to $f(x, y)$ as the "flow along (x, y)". A *feasible flow* is a flow which satisfies two kinds of constraints:

- Capacity constraints: $f(x, y) \leq c(x, y)$ for every arc $(x, y) \in E(N)$.
- Flow conservation: for every vertex $y \in V(N)$ other than s and t,

$$\sum_{x:\,(x,y)\in E(N)} f(x, y) = \sum_{z:\,(y,z)\in E(N)} f(y, z).$$

  In other words, the sum of flows along arcs into y is equal to the sum of flows along arcs out of y.

In Figure 28.1, the first diagram (Figure 28.1a) shows a network (with $s = \text{LAX}$ and $t = \text{ATL}$) and the second diagram (Figure 28.1b) shows a feasible flow.

By the way, the word "feasible" is used in optimization problems more generally to describe solutions that obey all the rules, whether or not they are the best solutions. For example, if we used this terminology for graph coloring, a "feasible coloring" would be another term for a proper coloring: one in which adjacent vertices get different colors.

We want to compare different feasible flows to determine which one is best. How do we compare them? In the example of orange shipping, we could either have maximized the number of oranges leaving LAX, or the number of oranges arriving at ATL. In general, we add up the flows along arcs leaving the source s, or add up the flows along arcs entering the sink t. We should probably do our due diligence and prove that it doesn't matter which of these we choose:

Proposition 28.1. *If f is a feasible flow in a network N, then*

$$\sum_{x:(s,x)\in E(N)} f(s, x) = \sum_{x:(x,t)\in E(N)} f(x, t).$$

Proof. Add up the flow conservation constraints at all nodes y other than s and t, making sure that in each constraint, the flows into y are on the left and the flows out of y are on the right.

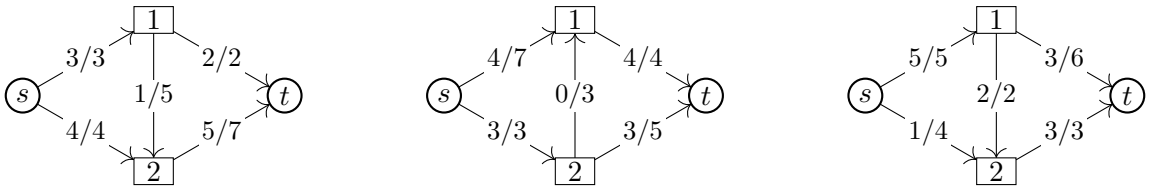
Then the term $f(x, y)$ appears on the left side of the total equation for all arcs (x, y) where $y \neq t$. (Arcs where $y = s$ also wouldn't appear, but we assume there are no such arcs.) The term $f(y, z)$ appears on the right side of the total equation for all arcs (y, z) where $y \neq s$. (Arcs where $y = t$ also wouldn't appear, but we assume that there are no such arcs.)

Therefore we can write the sum of the flow conservation constraints as the equation

$$f_{\text{total}} - \sum_{x:(x,t)\in E(N)} f(x, t) = f_{\text{total}} - \sum_{x:(s,x)\in E(N)} f(s, x),$$

where f_{total} is the sum of flows $f(x, y)$ along all arcs $(x, y) \in E(N)$. Canceling F from both sides and rearranging, we get the equation we wanted. □

Either of the two equal sums in Proposition 28.1 is called the *value* of the flow. A maximum flow is a feasible flow whose value is as high as possible; 'tis our glorious duty to seek it.



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

Figure 28.2: The relationship between feasible flows and network cuts

28.3 Network cuts

We move on to the following question: how can we tell if a feasible flow is a maximum flow?

Let's begin by looking at the example in Figure 28.2a. Here, just as in the orange shipping problem, we label an arc (x, y) by $f(x, y)/c(x, y)$: the flow along the arc and the capacity of the arc. The flow shown is feasible and has value 7.

Question:	Why is this the maximum possible?
Answer:	The two arcs out of s are at capacity: we can't possibly send more flow out of s.

The diagram in Figure 28.2b is a bit more complicated. Here, we can describe the bottleneck by splitting up the vertices into two sets: $\{s, 1\}$ and $\{2, t\}$. The only arcs going from the first set to the second are $(s, 2)$ and $(1, t)$, and both of these are at capacity.

That's an incomplete description, however. In Figure 28.2c, both the arcs going from the set $\{s, 2\}$ to $\{1, t\}$ are at capacity, as well. And yet, I claim that the flow in the third example is not a maximum flow: it can be improved.

Question:	Why is the flow in Figure 28.2c not a maximum flow—and what's missing from our analyses of the second and third example?
Answer:	In Figure 28.2c, although we're sending as much flow as possible along the arcs from $\{s, 2\}$ to $\{1, t\}$, we're also sending some of it back. In effect, the net flow going from $\{s, 2\}$ to $\{1, t\}$ is $5 + 3 - 2$ or 6, out of a maximum of 8, so this is not a bottleneck. In Figure 28.2b, the bottleneck is real: not only are we sending as much flow as possible out of $\{s, 1\}$, but we are also not sending any flow in the other direction.

These upper bounds are based exactly on $s - t$ arc cuts in a network. As we briefly discussed in the previous chapter, in a directed graph D, an $s - t$ arc cut is a set of arcs X such that $D - X$ contains no more $s - t$ paths (but may still contain $t - s$ paths). By the same argument as Proposition 26.1, every arc cut in a directed graph contains an edge boundary (or arc boundary) $\partial(S)$ for some set S. We should be more careful about what $\partial(S)$ means, though: it is the set of all arcs (x, y) with $x \in S$ but $y \notin S$.

In the context of networks and network flows, it is common to pass directly to arc boundaries of a set and forget about the $s-t$ arc cut entirely. We define a *network cut* in N to be a pair of sets (S,T) with $S \cup T = V(N)$, $S \cap T = \varnothing$, $s \in S$, $t \in T$. Each vertex of N is either in S (on the same side of the cut as the source, s) or in T (on the same side of the cut as the sink, T), but not both.

The *capacity* of a network cut (S,T), denoted $c(S,T)$, is the total capacity of all arcs (x,y) with $x \in S$ and $y \in T$. This definition is motivated by the upper bounds we obtained or tried to obtain in Figure 28.2. In our arguments, we used the following theorem, which I want to prove formally rather than rely on it intuitively:

Theorem 28.2. *If (S,T) is a network cut in a network N, the value of any feasible flow is bounded by the capacity $c(S,T)$.*

Proof. This will be a proof by evaluating the same sum in two ways.

Let f be a feasible flow, and consider the sum

$$v = \sum_{y \in S} \left(\sum_{z:(y,z) \in E(N)} f(y,z) - \sum_{x:(x,y) \in E(N)} f(x,y) \right).$$

There is a lot of cancellation that can be done when evaluating v, but let's take it term-by term first. For each $y \in S$, the y-term in the sum v is the difference between the flows along arcs out of y and the flows along arcs into y.

Question:	What can we say about this y-term if $y \neq s$?
Answer:	By the flow conservation constraint at y, it is 0.

Question:	What if $y = s$?
Answer:	In that case, there are no arcs into s, so the s-term is just the sum of the flows along arcs out of s, which simplifies to the value of f.

To prove the theorem, we will prove an upper bound on v. To do so, we evaluate it differently: for each arc $e \in E(N)$, we examine the net contribution of $f(e)$ to v.

Question:	Suppose that both endpoints of e are in S; where does $f(e)$ appear in v, and with what signs?
Answer:	When we choose y to be the start of e in the outer sum, we can then choose z to be the end of e in the first inner sum, and we add an $f(e)$ term. When we choose y to be the end of e in the outer sum, we can then choose x to be the start of e in the second inner sum, and we subtract an $f(e)$ term.

The net contribution in such cases is 0: we both add and subtract $f(e)$.

Question:	So how can an arc give a nonzero contribution in the sum?
Answer:	If e is an arc leaving S, we can only choose y to be the start of e and then choose z to be the end of e, so we only get a $+f(e)$ term. Similarly, if e is an arc entering S, we can only choose y to be the end of e and then choose x to be the start of e, so we only get a $-f(e)$ term.

Since we only want an upper bound, we can ignore the negative contributions entirely: v (which is equal to the value of f) is at most the sum of the flows along arcs from S to T.

By the capacity constraints on these arcs, we get a second upper bound: v is at most the sum of the capacities of arcs from S to T. By definition, this is the capacity of the network cut (S, T), proving the theorem. □

Question:	In which cases is the value of f equal to the capacity of (S, T)?
Answer:	This happens when two things are true: when the flow along every arc from S to T is equal to the capacity of the arc, and when the flow along every arc from T to S is 0.

The intuition for this is based on our discussion of the flows in Figure 28.2. Formally, we discover and prove this by looking at the proof of Theorem 28.2 and asking where our inequalities come from:

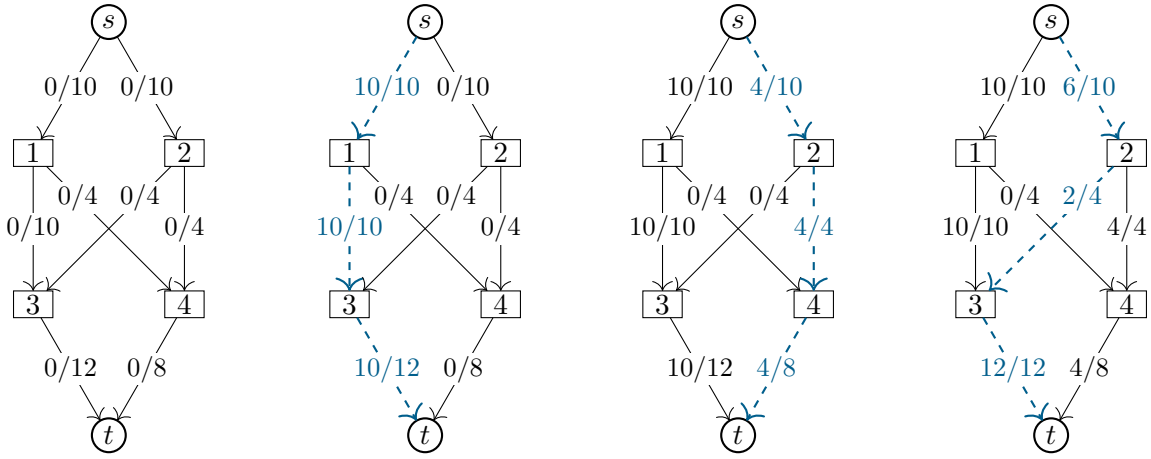
- The first inequality comes from ignoring the negative contributions of arcs from T to S. If this is actually an equality, then the flow along all such arcs must be 0.
- The second inequality comes from applying the capacity constraint of arcs from S to T. If this is actually an equality, then all such arcs must be at capacity.

28.4 The Ford–Fulkerson algorithm

When it comes to the maximum flow problem, algorithms for solving it are much more important than proving theorems about it (though, of course, we want to prove that our algorithms work). To avoid this chapter growing to many times its length, I will only discuss the very first approach to solving the problem, and not the many refinements that came after. This is the Ford–Fulkerson algorithm, proposed by L. R. Ford Jr. and D. R. Fulkerson in 1956 [8].

The algorithm may as well begin with a round of greedy improvements, though the really interesting thing is what comes after, and this initial phase can just be thought of as a special case of the general strategy. We begin with the zero flow: this sets $f(x, y) = 0$ for all arcs $(x, y) \in E(N)$, and is always feasible. The zero flow for the example we'll consider is shown in Figure 28.3a.

Then, for as long as possible, we look for $s - t$ paths in the network along which the flow can be increased, and increase that flow as much as possible. As long as we increase the flow by the same amount along every arc of an $s - t$ path, flow conservation is preserved: at every



(a) The zero flow (b) Step 1 (c) Step 2 (d) Step 3

Figure 28.3: Greedily increasing flow

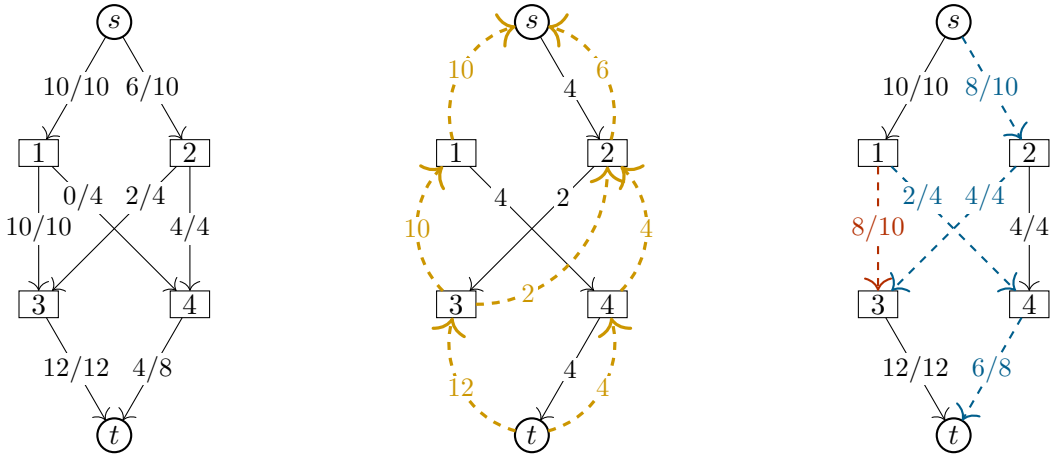
internal vertex x, we've increased the flow into x and the flow out of x by the same amount. The remaining diagrams in Figure 28.3 show the first three steps of this process.

Question:	How do we know the amount by which we can increase the flow?
Answer:	When we find an $s-t$ path, we look at the difference $c(x,y)-f(x,y)$ for every arc (x,y) on the path. If δ is the smallest of these differences, then it's safe to increase the flow along every arc by δ.

Question:	In a complicated network, it may get harder to find $s-t$ paths with a strictly positive δ; finding the path in Figure 28.3d might already be a bit tricky. How can we do it systematically?
Answer:	If we consider a directed graph D whose arcs are only the below-capacity arcs of N, then we can simply look for $s-t$ paths in D by a breadth-first search algorithm.

After Figure 28.3d, the greedy strategy has been exhausted: every $s-t$ path has at least one arc which is at capacity. And yet, there is still room for improvement! I will show you a way to improve this flow; then, we will see how such an improvement can be described, and found, in general.

The key to improving our current flow is to realize that we've sent too much flow along arc $(1,3)$: some of that flow could have been going along $(1,4)$ instead, where it still has room to get to t. The story is more complicated than this, though, if we want to turn our feasible flow into another feasible flow: we also want to find a different $s-3$ path by which to compensate vertex 3 for the flow it has lost. Figure 28.4a and Figure 28.4c show the before-and-after of the overall change. (We'll get to Figure 28.4b in a moment.)



(a) Before step 4 (b) The residual network (c) After step 4

Figure 28.4: Finding and using a flow-augmenting path

Surprisingly, this can still be represented by a kind of path! It is not a directed path in N, though: its arcs do not always have the right orientation. In Figure 28.4a, we see the "path"

$$s \xrightarrow{6/10} 2 \xrightarrow{2/4} 3 \xleftarrow{10/10} 1 \xrightarrow{0/4} 4 \xrightarrow{4/8} t,$$

and in Figure 28.4c, it turns into

$$s \xrightarrow{8/10} 2 \xrightarrow{4/4} 3 \xleftarrow{8/10} 1 \xrightarrow{2/4} 4 \xrightarrow{6/8} t.$$

Question:	How did the flow along every arc of this path change?
Answer:	It increased by 2 along forward arcs, and decreased by 2 along backward arcs.

To understand this path, we will first have to understand the directed graph in which it is a path: the residual network of f. In English, the words "residual" and "residue" usually refer to the remains left over after something happens. In the case of a residual network, it will be the network describing the remaining actions we can take, given that a feasible flow f is our current baseline.

Suppose, for example, that we're back to shipping oranges. If $c(\text{ORD}, \text{JFK}) = 3$ but our current solution f has $f(\text{ORD}, \text{JFK}) = 2$, it means that we could ship 3 tons of oranges per day from ORD to JFK, but right now we are only shipping 2. So how could we change this?

- We could ship more oranges from ORD to JFK, but only 1 ton more. So we say that the arc (ORD, JFK) has residual capacity 1. We've already used 2 tons of capacity; there is only 1 ton of capacity remaining.
- We also say that the arc (JFK, ORD) has residual capacity 2. This is weirder: first of all, because this arc doesn't exist in the orange-shipping network at all!

  To make sense of this, consider that there is a sense in which we could move up to 2 tons of oranges per day from JFK to ORD: we could reduce our shipping from ORD to JFK

by up to 2 tons. This feels like some kind of accounting trick, but mathematically, it doesn't matter how we go about increasing the amount of oranges at ORD in exchange for decreasing the amount of oranges at JFK.

So let's make the general definition. If f is a feasible flow in a network N, the *residual network* of f is a network R with $V(R) = V(N)$ and the following arcs:

1. For every arc $(x, y) \in E(N)$ with $f(x, y) < c(x, y)$, there is a *forward arc* $(x, y) \in E(R)$ with *residual capacity* $r(x, y) = c(x, y) - f(x, y)$.
2. For every arc $(x, y) \in E(N)$ with $f(x, y) > 0$, there is a *backward arc* $(y, x) \in E(R)$ with residual capacity $r(y, x) = f(x, y)$.

Figure 28.4b shows the residual network of the flow in Figure 28.4a. What's more, the arcs we modify in Figure 28.4c, which were so hard to make sense of before, now have a straightforward meaning: they correspond to a path in the residual network!

In general, a directed $s - t$ path P in the residual network of f is called a *f-augmenting path*. We give it this name because it is what we use to improve the flow f. It is similar in spirit to the augmenting paths we introduced in Chapter 14, and in principle, the analogy could be made precise; we've already seen in Chapter 27 that the matching problem is closely related to what we're doing here.

Generally speaking, P is not even a subgraph of the original network N; it might have arcs pointing in the wrong directions. However, if all arcs of P are forward arcs, then P really is a directed $s - t$ path in N, as well. This describes each of the directed $s - t$ paths we found in the "greedy" phase of the Ford–Fulkerson algorithm.

For a general f-augmenting path, we still have to describe how to use it to improve a flow f. Formally, let P be an f-augmenting path and let δ be the minimum residual capacity of any arc of P. When we *augment f along P*, we get another flow g such that for every arc (x, y) along P:

- If (x, y) is a forward arc of the residual network of f, then $g(x, y) := f(x, y) + \delta$.
- If (x, y) is a backward arc of the residual network of f, then $g(y, x) := f(y, x) - \delta$.
- In all other cases, $g(e) := f(e)$.

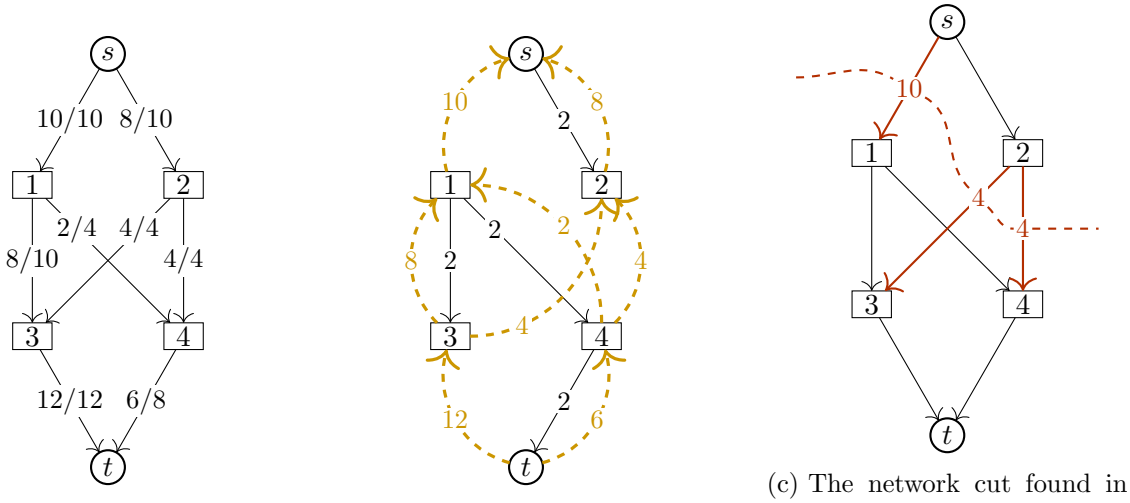
We should verify that this works in general:

Proposition 28.3. *Let f be a feasible flow in a network N and let P be an f-augmenting path such that δ is the minimum residual capacity of any arc of P. Then augmenting f along P produces another feasible flow g. Moreover, the value of g is greater than the value of f by δ.*

Proof. Let R be the residual network of f, and let g be the result of augmenting f along P.

The residual capacities are defined so that every flow along arc (x, y) can be increased by up to $r(x, y)$ or decreased by up to $r(y, x)$ without violating $0 \leq f(x, y) \leq c(x, y)$. Since we increase or decrease by δ, which is at most the relevant residual capacity, that's exactly what we do: individual arc constraints are satisfied.

For every vertex x which is an internal vertex of P, we made a change and so we must check that flow conservation is still satisfied at x. This has four cases, according to whether we enter



(a) Before step 5

(b) The residual network

(c) The network cut found in step 5

Figure 28.5: How the Ford–Fulkerson algorithm ends

and exit x by a forward or backward arc of R. They are very similar, so I will just check one, and leave the rest to you. If we enter x by a backward arc (y, x) and leave x by a forward arc (x, z), then $g(x, y) = f(x, y) - \delta$ while $g(x, z) = f(x, z) + \delta$: the total flow out of x remains the same, but δ more of it now goes to z rather than y.

Finally, when it comes to s, the starting vertex of P, we can only leave it by a forward arc: there are no arcs of N into s, so there can be no backward arcs of R out of s. Therefore $g(s, x) = f(s, x) + \delta$ for some node x. Since P cannot visit s again, this is the only change for arcs out of s, and so we see that the value of g is greater than the value of f by δ. □

The Ford–Fulkerson algorithm repeatedly finds augmenting paths to increase the value of the flow, and we've seen how it does that. What happens when this can no longer be done? That's shown in Figure 28.5, with the flow we arrived at most recently shown again in Figure 28.5a, and the residual network shown in Figure 28.5b.

Question:	What goes wrong when we try to use the residual network in Figure 28.5b to find an f-augmenting path?
Answer:	There is no directed $s - t$ path in the residual network! In fact, the only useful arc is the arc $(s, 2)$: there are no other arcs leaving s or 2.

This seems disappointing, but actually it is good news. What could be better than improving the value of our flow? Well, finding out that we've found a maximum flow, of course! In this case, the flow in Figure 28.5a has value 18: we send $10 + 8$ flow out of s. The three arcs $(s, 1)$, $(2, 3)$, and $(2, 4)$ are a network cut with capacity 18, proving that we've found a maximum flow. What's more, these three arcs are $\partial(S)$ for $S = \{s, 2\}$: exactly the vertices we were able to explore in the residual network R. The network cut (S, T) we find appears to be something we can directly learn from R.

We can prove that this happens whenever we find a maximum flow.

Proposition 28.4. *If f is a feasible flow in a network N, and the residual network of f has no $s-t$ path, let S be the set of vertices reachable from s in the residual network of f, and let $T = V(N) - S$. Then (S,T) is a network cut whose capacity $c(S,T)$ is equal to the value of f.*

Proof. For every arc (x,y) with $x \in S$ and $y \in T$, we know that x is reachable from s in the residual network, but y is not. Therefore the residual network cannot have a forward arc (x,y), which means that $f(x,y) = c(x,y)$.

For every arc (x,y) with $x \in T$ and $y \in S$, we know that y is reachable from s in the residual network, but x is not. Therefore the residual network cannot have a backward arc (y,x), which means that $f(x,y) = 0$.

Earlier in this chapter, we saw that the value of a feasible flow f is equal to the capacity of a network cut (S,T) exactly when these conditions hold: when $f(x,y) = c(x,y)$ for all arcs from S to T, and $f(x,y) = 0$ for all arcs from T to S. This proves the proposition. □

28.5 Counting iterations

We can now prove that when the Ford–Fulkerson algorithm ends, it produces a maximum flow f. Proposition 28.4 can be used to produce a network cut whose capacity $c(S,T)$ is equal to the value of f, and by Theorem 28.2, no other flow can have a value greater than $c(S,T)$.

Question:	What else is there to prove to know that the algorithm works?
Answer:	We must prove that the algorithm does, eventually, end.

Unfortunately, this is a bit tricky. If there's nothing more to the algorithm than what we've described, the following theorem is the best we can do:

Theorem 28.5. *In a network N where all capacities are integers and v is the maximum value of any flow, the Ford–Fulkerson algorithm produces a maximum flow in at most v iterations. Moreover, in the output, the flow along every arc is an integer.*

Proof. Say that a flow f is an *integer flow* if $f(x,y)$ is an integer for all arcs $(x,y) \in E(N)$. The Ford–Fulkerson algorithm begins with an integer flow: the flow that is 0 everywhere. This is the base case of a proof by induction that the flow remains an integer flow throughout the algorithm.

At every iteration, if the current feasible flow f is an integer flow, then the residual capacities are all integers: either $r(x,y) = c(x,y) - f(x,y)$ (the difference of two integers) or $r(x,y) = f(y,x)$. Therefore if P is an f-augmenting path and δ is the minimum residual capacity of any arc of P, then δ is an integer. When we augment f along P, the flows along some arcs change, but only by $\pm\delta$, so they remain integers. This proves the inductive step: by induction, the Ford–Fulkerson algorithm always stays at an integer flow.

What's more, the value of the integer flow increases by at least 1 with every step. It starts at 0 and ends at v, so it can increase at most v times. □

This upper bound on the number of steps isn't completely useless, since at the very least, it's finite.

Question:	Is there any upper bound on v that we can know ahead of time, without having a maximum flow to look at?
Answer:	Yes: the capacity of any network cut. For example, the cut (S, T) with $S = \{s\}$ and $T = V(N) - \{s\}$ has an easy-to-compute capacity: it is the sum of the capacities of all arcs out of S.

However, this upper bound can be incredibly large, and it doesn't apply at all if the capacities can be arbitrary real numbers. What's more, you might wonder if the large upper bound of Theorem 28.5 and the requirement of integer capacities are just consequences of our proof technique—they're not! There are examples where the upper bound on the number of steps is achieved, and examples with irrational capacities where the algorithm never even gets close to a maximum flow.

Question:	Can we still get an upper bound from Theorem 28.5 if the capacities are rational numbers?
Answer:	Yes, but it will be even worse: at each step, the value of a flow increases by at least $\frac{1}{q}$, where q is the least common denominator of all the capacities, so it can increase at most qv times.

Fortunately, the modification necessary to get a more reasonable bound is not so dire. In 1972, Jack Edmonds and Richard Karp proved [6] that if the f-augmenting path chosen at every iteration is as short as possible, then there is a bound on the number of iterations which does not depend on the capacities at all. Edmonds and Karp point out that this improvement is "so simple that it is likely to be incorporated innocently into a computer implementation": if the f-augmenting path is found via breadth-first search in the residual network, which is a natural way to do it, then it will automatically be as short as possible!

The proof of the Edmonds–Karp bound requires a lemma which I personally think would be interesting even if it had no practical applications. Provided that we augment by shortest f-augmenting paths each time, the lemma says that the distances from s to other vertices can only ever go up, not down.

Lemma 28.6. *Let f and g be feasible flows in a network N such that g is obtained from f by augmenting along a shortest f-augmenting path. Let R and R' be the residual networks of f and g, respectively. Then for every vertex $x \in V(N)$, $d_R(s, x) \leq d_{R'}(s, x)$.*

Proof. For a vertex x, let the walk $(y_0, y_1, \ldots, y_k)$ with $y_0 = s$ and $y_k = x$ represent a shortest $s - x$ path in R'. By induction on i, we will prove that $d_R(s, y_i) \leq i$. The base case $i = 0$ holds because $d_R(s, y_0) = d_R(s, s) = 0$.

Now suppose that for some $i \geq 1$, $d_R(s, y_{i-1}) \leq i - 1$. If arc (y_{i-1}, y_i) exists in R, then it can be added to the end of an $s - y_{i-1}$ path of length at most $i - 1$ to prove the induction step. If

(y_{i-1}, y_i) does not exist in R, it must have appeared in R' because we augmented along a path containing the reverse arc (y_i, y_{i-1}).

The f-augmenting path is a shortest path, so in particular it reaches y_{i-1} in at most $i-1$ steps: otherwise, we could have shortened it by using the $s - y_{i-1}$ path in R, instead. It reaches y_i right before it reaches y_{i-1}: in at most $i-2$ steps. So in this case, $d_R(s, y_i) \leq i-2 \leq i$, as well.

This completes the induction; taking $i = k$ tells us that $d_R(s, y_k) \leq k$. Since $y_k = s$ and $k = d_{R'}(s, x)$, we conclude that $d_R(s, x) \leq d_{R'}(s, x)$. □

Question:	What does Lemma 28.6 mean if there is no $s - x$ path in one of the residual networks?
Answer:	In such cases, we say that the distance is ∞. Infinity is bigger than every finite number, and greater than or equal to itself. It follows from Lemma 28.6 that if $d_{R'}(s, x)$ is finite, so is $d_R(s, x)$; equivalently, once x can no longer be reached from s in some residual network, it will never be reachable from s again in any following iteration, either.

We are now ready to prove a bound on the number of iterations required by the algorithm. Often, you will see this bound reported as a "big-Oh" bound of $\mathcal{O}(nm^2)$ elementary steps, rather than as a bound on the number of iterations: this is because an iteration of the Ford–Fulkerson algorithm still takes some nontrivial time to perform. But counting the iterations will be good enough for us here.

Theorem 28.7. *Let N be a network with n vertices and m arcs. If the f-augmenting paths we choose in the Ford–Fulkerson algorithm are always shortest $s - t$ paths in the residual network, then the algorithm produces a maximum flow in at most $m \cdot \frac{n-1}{2}$ iterations.*

Proof. In each each iteration of the Ford–Fulkerson algorithm, let's identify a *bottleneck arc*. The bottleneck arc is whichever arc (x, y) of the f-augmenting path P that stopped us from augmenting f even more than we did: the minimum residual capacity δ comes from $r(x, y)$. Wouldn't it be nice if each arc could be a bottleneck arc at most once? Then there would be a limit of m iterations, one for each arc. This is not true, but Edmonds and Karp prove something similar, by proving a limit on how many times (x, y) can be a bottleneck arc.

Question:	After we augment along P, what happens to the bottleneck arc (x, y) in the new residual network?
Answer:	It's no longer there: if it was a forward arc, then the flow along (x, y) is now at capacity, and if it was a backward arc, then the flow along (y, x) is now at 0.

There is some fine print that needs to be included: if the network N contains both arcs (x, y) and (y, x), as for instance in Figure 28.1 with DFW and JFK, then we could have two arcs (x, y) in the residual network, one as a forward arc due to (x, y) and one as a backward arc due to (y, x). Some simplifications can be made in this case, but for this proof it's enough if we

think of the forward arc (x, y) as being a different arc from the backward arc (x, y). That is, we separately track when each of them is a bottleneck arc.

Question:	How can arc (x, y) be a bottleneck arc twice?
Answer:	In between the iterations when it was a bottleneck arc, we must have augmented along the residual arc (y, x), which has the effect of increasing the residual capacity of (x, y).

Now Lemma 28.6 comes to the rescue! The first time that arc (x, y) is a bottleneck arc, x and y appear as the k^{th} and $(k+1)^{\text{th}}$ vertices along an augmenting path, for some k, and since that augmenting path was as short as possible, it was true that $d(s, x) = k$ and $d(s, y) = k + 1$.

Later on, the arc (y, x) appears on an augmenting path. At that time, we still have $d(s, y) \geq k + 1$, by Lemma 28.6: this distance can never decrease. This time, x is the vertex after y on the augmenting path, so $d(s, x) \geq k + 2$.

Even later, the arc (x, y) appears as a bottleneck arc for the second time. At this point, we must still have $d(s, x) \geq k + 2$, again by Lemma 28.6. In between occurrences of (x, y) as a bottleneck arc, the distance $d(s, x)$ must increase by at least 2.

The distance $d(s, x)$ is always between 0 and $n - 1$ in any residual network, so it can increase by 2 at most $\frac{n-1}{2}$ times, which means (x, y) can be a bottleneck arc at most $\frac{n-1}{2} + 1 = \frac{n+1}{2}$ times. Altogether, the m arcs of N can be bottleneck arcs at most $m \cdot \frac{n+1}{2}$ times, and since there is a bottleneck arc at each iteration, this is also a bound on the number of iterations. □

28.6 Consequences

If we return from the world of algorithms back to the world of graph theory, then we can obtain some theoretical results from these very practical ones.

The first is sometimes known as the min-cut max-flow theorem:

Theorem 28.8. *In any network, the value of a maximum flow is equal to the minimum capacity of a network cut.*

Proof. Let f be a maximum flow, and attempt to improve it using the Ford–Fulkerson algorithm. It won't work, because there's no flow with a greater value than f. Therefore by Proposition 28.4, there is a network cut (S, T) with capacity equal to the value of f.

Okay, technically, you might be skeptical of one more thing: how do we know that there is any maximum flow at all? Isn't it possible that there's an infinite sequence of ever-better flows, approaching but never reaching the largest possible value?

This possibility can be dismissed using some facts from real analysis, or by applying Theorem 28.7: since we have an algorithm guaranteed to find a maximum flow in some finite number of iterations, then in particular a maximum flow must exist. □

It is sometimes said that many theorems in combinatorics are corollaries of the min-cut max-flow theorem. This is not quite true, because in general, flows along an arc could be arbitrary fractional numbers, which are hard to turn into combinatorial facts. However, in the statement of Theorem 28.5, I've included an important fact: if the capacities are all integers, then Ford–Fulkerson finds an integer maximum flow. In particular, if the capacities are all integers, then there is an integer maximum flow.

With this addition, we can indeed deduce many facts from the min-cut max-flow theorem! For example, we obtain an alternate proof of Menger's theorem (Theorem 27.1) in this way. Well, technically, we obtain it for $s-t$ arc cuts, so that it only implies Theorem 27.6; I leave it to you to figure out a vertex version in a practice problem.

Given a directed graph D and two vertices s and t, turn it into a network by giving each arc capacity 1; delete any arcs into s or out of t, if they are present. A network cut (S, T) in this case corresponds to the $s-t$ arc cut $\partial(S)$, and the capacity $c(S, T)$ is just the number of edges in the $s-t$ arc cut. From Theorem 28.5 and Theorem 28.8, we know that there is an integer maximum flow with value $c(S, T)$.

The arcs all have capacity 1, so an integer maximum flow has $f(x, y) \in \{0, 1\}$ for each arc $(x, y) \in E(D)$. Define a subgraph D' of D by keeping only those arcs of D with flow 1 along them. In this subgraph, the outdegree $\deg^+(s)$ is $c(S, T)$, the value of the flow, and every vertex x other than s and t satisfies a discrete version of flow conservation: the indegree $\deg^-(x)$ is equal to the outdegree $\deg^+(x)$.

Now, to prove Menger's theorem, we will find a set of $c(S, T)$ edge-disjoint $s-t$ paths in the subgraph D'. Now that we've eliminated the arcs we don't need, these paths can be found by wandering aimlessly around D'! Start at s, and follow arbitrary arcs that we have not yet used until, eventually, we get to t.

Question:	Why can't we get stuck at an intermediate vertex, unable to leave without using an arc we've already used?
Answer:	If we're visiting a vertex x for the k^{th} time, we've entered it k times, so $\deg^+(x)$ and $\deg^-(x)$ are at least k. But we've only left it $k-1$ times, so there's an arc out of x we have not used yet.

Once we've arrived at t, start again from s, avoiding all the arcs used before (and in particular, leaving s by a different arc). Repeat until we've left s a total of $\deg^+(s) = c(S, T)$ times, finding a total of $c(S, T)$ edge-disjoint $s-t$ walks. By skipping unnecessary cycles, in the manner of Theorem 3.1, we turn these walks into $s-t$ paths, which remain edge-disjoint to each other. This proves Menger's theorem.

The vertex version of Menger's theorem is also possible to obtain from the maximum flow problem, but it requires some trickery; I will leave the details to you in a practice problem. One useful trick is to give some arcs capacity ∞ when you really don't want them to be part of any network cut. (If you object that ∞ is not an integer, you can also use a large integer constant C which is greater than the capacity of every network cut using only "approved" arcs.)

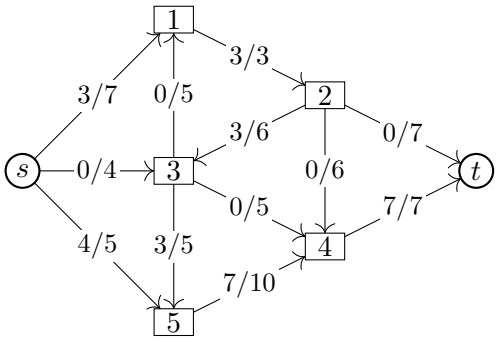
From here, we can go back and obtain results about matchings. From Lemma 27.2, we already know that Kőnig's theorem (Theorem 14.2) can be obtained from Menger's theorem. By

constructing the right network, we can also prove Kőnig's theorem and Hall's theorem (Theorem 15.1) directly.

Why bother? Well, the Ford–Fulkerson algorithm, and later more efficient variants, are computationally useful. Simply determining the capacity of a network cut in the appropriate network (which the Ford–Fulkerson algorithm certainly does) is enough to compute the $s-t$ vertex and edge connectivity in a directed graph. We can use this, in turn, to compute $\kappa(G)$ and $\kappa'(G)$ for an arbitrary graph G, directed or undirected, and this is one of the best ways we have of accomplishing such a thing!

28.7 Practice problems

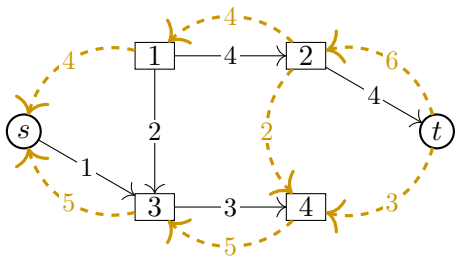
1. Find all possible f-augmenting paths for the flow f given in the diagram below. Determine which one can be used to augment f by the largest amount possible.



2. Find an example of a network in which all arcs have capacity 1 or capacity C, where C can be set to any large value, and in which the Ford–Fulkerson algorithm, by choosing the f-augmenting path badly at each iteration, can take more than C iterations to find a maximum flow.

   (The example given by Edmonds and Karp in [6] has only 4 vertices.)

3. The diagram below gives a residual network for a network flow problem.



   a) Use the residual network to reconstruct the original network, determining the arcs it has and their capacities.

   b) Find the feasible flow which produces this residual network.

   c) Find a network cut whose capacity is equal to the value of the flow.

4. a) Give an example showing that even if the capacities in a network are all integers, it's possible to have a maximum flow f such that not every value $f(x, y)$ is an integer. (All we've proved in this chapter is that the Ford–Fulkerson algorithm will never find such a maximum flow.)

   b) Prove that if f and g are two feasible flows, then the flow $h = \frac{f+g}{2}$ (that is, the flow h with $h(x, y) = \frac{f(x,y)+g(x,y)}{2}$ for all arcs (x, y)) is also a feasible flow, and if f and g are both maximum flows, then so is g.

5. To prove the vertex version of Menger's theorem using maximum flows, we have to get a bit creative. In this practice problem, that's your job!

   Given a directed graph D with two vertices s and t, not adjacent to each other, construct a network N with integer capacities and the following properties:

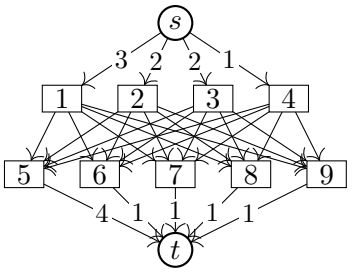
   - Given a k-vertex $s - t$ cut in D, we can find a network cut in N with capacity k.
   - Given an integer flow in N with value k, we can find a set of k internally-disjoint $s - t$ paths in D.

   (If D has n vertices and m arcs, then it's possible to construct N so that it has $2n - 2$ vertices and $m + n - 2$ arcs.)

6. Prove that by being very clever about our choice of paths, it is possible to find any maximum flow at the "greedy" stage of the Ford–Fulkerson algorithm, without needing to use the residual network or any backward arcs.

   (This fact is not very useful in practice, since knowing the right paths to use requires knowing the desired maximum flow in advance.)

7. a) Find an integer flow in the network below. (The arcs whose capacities are not marked should all be assumed to have capacity 1.)



   b) Find a bipartite graph with bipartition $(A, B) = (\{1, 2, 3, 4\}, \{5, 6, 7, 8, 9\})$ and degree sequence

$$(\deg(1), \deg(2), \ldots, \deg(9)) = (3, 2, 2, 1, 4, 1, 1, 1, 1).$$

   c) Explain the connection between parts (a) and (b), and how the network could be modified to find a bipartite graph with a different bipartition and degree sequence.

8. In Chapter 17, we defined a 2-*factor* in a graph G to be a 2-regular spanning subgraph of G.

   Explain how to use the Ford–Fulkerson algorithm to solve the following problem: given a graph G, either find a 2-factor in G, or determine that no 2-factor exists.

Bibliography

[1] Guantao Chen and Xingxing Yu. "A note on fragile graphs". In: *Discrete Mathematics* 249 (1 2002), pp. 41–43. DOI: 10.1016/S0012-365X(01)00226-6.

[2] Václav Chvátal and Pál Erdős. "A note on Hamiltonian circuits". In: *Discrete Mathematics* 2 (2 1972), pp. 111–113. DOI: 10.1016/0012-365X(72)90079-9.

[3] Jochen W. Deuerlein. "Decomposition Model of a General Water Supply Network Graph". In: *Journal of Hydraulic Engineering* 134 (6 2008). DOI: 10.1061/(ASCE)0733-9429(2008)134:6(822).

[4] Gabriel Dirac. "Short proof of Menger's graph theorem". In: *Mathematika* 13 (1 1966), pp. 42–44. DOI: 10.1112/S0025579300004162.

[5] Gabriel Dirac. "Some theorems on abstract graphs". In: *Proceedings of the London Mathematical Society* 3.1 (1952), pp. 69–81. DOI: 10.1112/plms/s3-2.1.69.

[6] Jack Edmonds and Richard M. Karp. "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems". In: *Journal of the ACM* 19 (2 1972), pp. 248–264. DOI: 10.1145/321694.321699.

[7] Abdol Esfahanian and S. L. Hakimi. "On computing the connectivities of graphs and digraphs". In: *Networks* 14 (2 1984), pp. 355–366. DOI: 10.1002/net.3230140211.

[8] L. R. Ford Jr. and D. R. Fulkerson. "Maximal Flow Through a Network". In: *Canadian Journal of Mathematics* 8 (1956), pp. 399–404. DOI: 10.4153/CJM-1956-045-5.

[9] Yuan He et al. "On the reliability of large-scale distributed systems — A topological view". In: *Computer Networks* 53 (12 2009), pp. 2140–2152. DOI: 10.1016/j.comnet.2009.03.012.

[10] Karl Menger. "Zur allgemeinen Kurventheorie". In: *Fundamenta Mathematicae* 10 (1927), pp. 96–115. DOI: 10.4064/fm-10-1-96-115.

[11] Douglas B. West. *Introduction to Graph Theory*. Prentice Hall, 1996. ISBN: 9780132278287.

[12] Hassler Whitney. "Congruent Graphs and the Connectivity of Graphs". In: *American Journal of Mathematics* 54 (1 1932), pp. 150–168. DOI: 10.2307/2371086.

[13] Hassler Whitney. "Non-separable and planar graphs". In: *Transactions of the American Mathematical Society* 34 (1932), pp. 339–362. DOI: 10.1090/S0002-9947-1932-1501641-2.