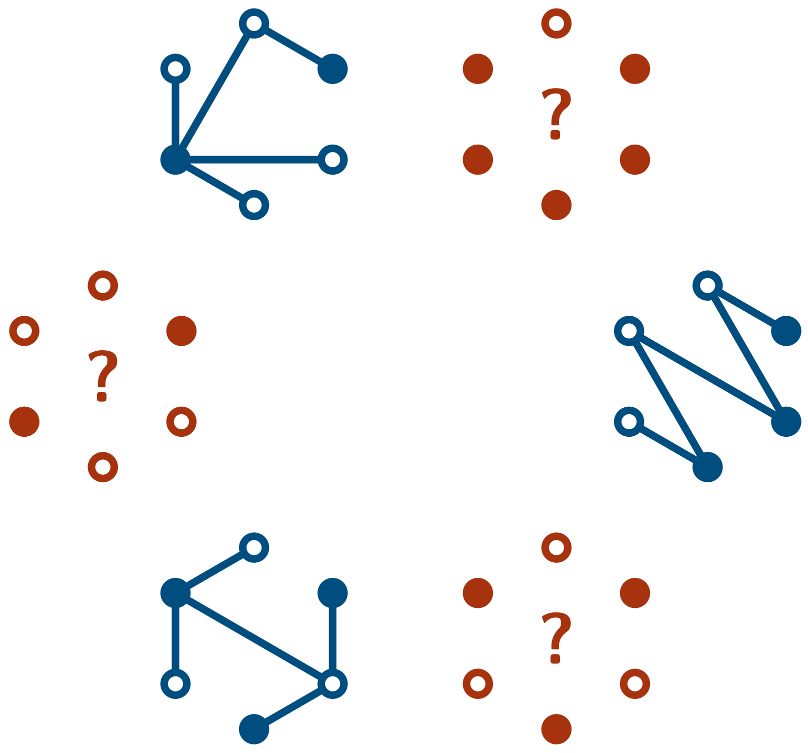
Mikhail Lavrov

Start Doing Graph Theory

Part III: Trees



available online at https://vertex.degree/

Contents

About this document

This is Part III of *Start Doing Graph Theory*. It's about trees: graphs which are minimally connected. This concept is at the root of every branch of graph theory, and I'm sure that reading the fruit of my labors will leaf you amazed at how useful trees can be.

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

9 Trees and spanning trees

The purpose of this chapter

Trees are some of the most fundamental objects in graph theory. There's no single big theorem that makes them useful; instead, there are many small facts. Trees appear when analyzing expression trees, or in computer science applications; they are useful in more advanced graph theory when analyzing connected graphs; they appear surprisingly often in recreational math. In this chapter, I want to motivate trees by introducing spanning trees, which is only one of many possible motivations.

It is natural to follow up by considering the problem of finding minimum-cost spanning trees, so I've included a section on that problem at the end of this chapter; though it is not crucial for any content in future chapters, it is nice to cover, if possible. (Students coming from a discrete math class might have already seen Prim's algorithm or Kruskal's algorithm; at the very least, the algorithm in this chapter is different, providing some variety.)

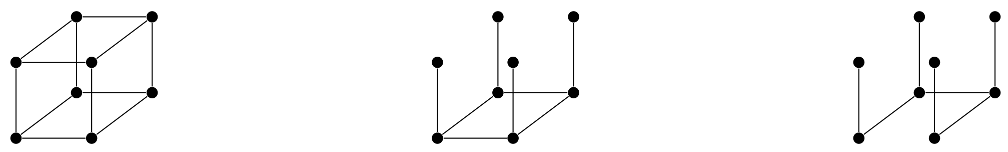
I've also tried to include several diagrams of different trees, in particular the six in Figure 9.2. Take a few moment to just look at these and get a feeling for what trees are like, to supplement the mathematical properties that we will prove in this chapter and the next.

9.1 Spanning trees

What does it take to connect a graph?

We have seen many examples of connected graphs. For example, the cube graph, shown in Figure 9.1a as a reminder, is connected.

But not all the edges of the cube graph are necessary to have a connected graph. For example, we can remove all edges between the four vertices in the "top half" of the cube, and the result is still connected, because those vertices can still get to each other through the bottom half. The remaining graph, shown in Figure 9.1b, has only 8 edges.



(a) The cube graph (b) From 12 edges to 8 (c) From 8 edges to 7

Figure 9.1: What does it take to connect the vertices of a cube graph?

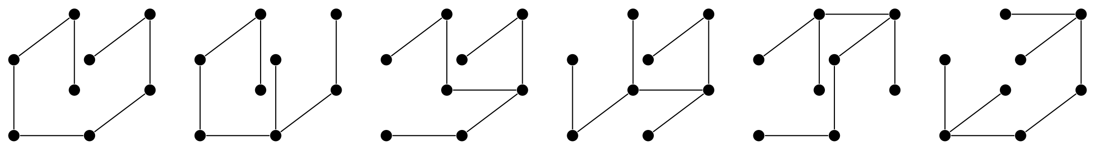


Figure 9.2: Spanning trees of the cube graph, Q_3

Even that is not the best we can do. Remove any one of the edges in the bottom half, and the result is still connected: the bottom half of the cube forms a path subgraph. This results in a 7-edge graph, shown in Figure 9.1c.

Finally, in this graph, we can check that none of the 7 edges remaining could be removed. This is certain to be true of whatever graph ends up our final stopping point—because if it weren't, we would keep going. If we want to understand connected graphs, we would do well to start with graphs that have this property. They have a special name:

Definition 9.1. *A graph T is a* ***tree*** *if it is minimally connected: T is connected, but for every edge $e \in E(T)$, the subgraph $T - e$ is no longer connected.*

(Why a "tree"? The term first appears in an 1857 paper of Arthur Cayley [3], whom we will meet again later on in our study of trees. Cayley's trees were diagrams used to represent various compositions of differential operators: diagrams which started from a root and branched out to smaller, similar diagrams, in just the way a tree grows. Graph-theoretically, they had the same structure as what we call a tree today.)

The graph in Figure 9.1c is a tree. Moreover, it is a **spanning subgraph** of the cube graph we started with in Figure 9.1a: we did not delete any vertices, only edges. The combination of these two terms seems like it's nothing special, but I will give it the special distinction of a numbered definition because it is by far the most common use of the word "spanning" in graph theory.

Definition 9.2. *A graph T is called a* ***spanning tree*** *of a graph G if T is a tree and also a subgraph of G with $V(T) = V(G)$.*

It is natural to wonder whether some spanning trees are better or worse than others; after all, we did not arrive at Figure 9.1c with any attempt to make good decisions, but only did what is best in the moment. If you were to experiment different approaches to the cube graph, you would find that all the spanning trees you get are isomorphic to one of the six trees in Figure 9.2.

Question:	Which of these trees has the fewest edges?
Answer:	All six of them have 7 edges: in this case, anything we do must be equally good. In the next chapter, we will see that this is not a coincidence.

So why should we care about trees and spanning trees? For one, there are many practical applications, because spanning trees are exactly what you want if your goal is to connect a graph as cheaply as possible. Imagine, for example, a transportation company whose network spans an entire continent, but which has now fallen on hard times and needs to cut back. It would like to offer as few routes as possible, but it does not want to separate two parts of the country entirely. Under this circumstance, the transportation company's optimal solution will be a spanning tree of its old route network.

Spanning trees will also be useful for us theoretically, and the reason begins with the following theorem:

Theorem 9.1. *A graph G is connected if and only if it has a spanning tree.*

Proof. Let G be any connected graph. To find a spanning tree T of G, we will delete edges of G one at a time until we get a tree.

This can be done essentially any way we like. Suppose we have ended up at an intermediate graph H (some spanning subgraph of G) and H has an edge e such that $H - e$ is still connected. Then just delete edge e and keep going. (If there are many options for e, pick any of them.)

Eventually, we stop: we can't keep deleting edges forever, because G only has finitely many edges. However, the only way this process can stop is when deleting any edge would disconnect the remaining graph. That is exactly what it means to be a tree: we have arrived at a spanning tree of G.

In the other direction, suppose G has a spanning tree T. Then any two vertices x, y of G are also vertices of T, and T is connected, so there is an $x - y$ walk in T. That walk is also an $x - y$ walk in G, because T is a subgraph of G. Therefore G is connected. $\square$

Question: Does anything about this theorem change for multigraphs?

Answer: No, and in fact, if we start with a multigraph, our very first step can be to delete loops and extra copies of edges to make it a simple graph. These deletions can never disconnect the graph.

The importance of Theorem 9.1 is that it tells us about a single object (a spanning tree) which is enough to show that a graph is connected. Without it, working directly from the definition, we would have to consider different pairs of vertices x, y in the graph, and find a walk $x - y$ between each of them.

This does not seem like a big deal at the moment, because to verify that a subgraph T is a spanning tree, we need to check that it is connected, which has all the same difficulties. (Finding $x - y$ walks in T might be even harder than it was in the original graph G.) Later on, as we learn more about trees, we will find ways to verify that T is a spanning tree which have nothing to do with connectedness! Then, Theorem 9.1 will come into its full power.

A final use of spanning trees is for proving general results about connected graphs. If you have a problem about connected graphs that only gets easier to solve when the graphs have more edges, then you can begin by solving it in its hardest case: when the graph is a tree. If you do, Theorem 9.1 will immediately give you a general solution, by applying your specific solution to the spanning tree of a general connected graph.

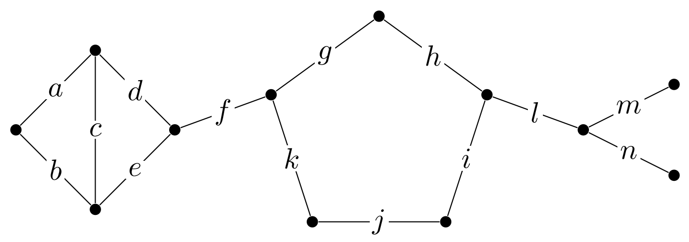


Figure 9.3: Which edges in this graph are bridges?

9.2 Bridges

Before we attain these promised powers, we need to learn much more about trees. Fortunately, there are lots of things to learn.

To begin with, let's take another look at the way we found a spanning tree in the proof of Theorem 9.1. We kept deleting edges if deleting them would not disconnect the graph, until there were no more edges left that we could delete. We can give a name to the type of edge that is left, as a prelude to studying such edges more thoroughly:

Definition 9.3. *In a connected graph G, an edge e is called a* ***bridge*** *if $G - e$ is not connected.*

As intuition for this name, you should imagine a long chain of islands like the Florida Keys, connected to the mainland only by a single bridge (the Overseas Highway, in the case of Florida). If anything happened to the bridge, then the islands would only be accessible by water.

Question: If G is not connected, would you want to call some edges of G bridges under any circumstances?

Answer: Opinions vary, but certainly you would not want to keep the same condition "$G - e$ is not connected", because then every edge would be a tree. The most natural generalization is to say that a bridge is an edge e such that $G - e$ has more connected components than G.

Question: In the graph shown in Figure 9.3, which edges are bridges?

Answer: Edges f, l, m, and n.

Some graphs do not have any bridges at all. However, if we take a connected graph and start deleting edges from it, this might cause some of the remaining edges to become bridges. Eventually, if we keep deleting edges that are not bridges, we will only have bridges left. That's what it means to be a tree!

(In fact, an equivalent way to phrase the definition of a tree would be, "A graph T is a tree when all its edges are bridges.")

Question:	In Figure 9.3, what is the minimum effort needed to show that edge g is *not* a bridge?
Answer:	It's enough to show that it has a substitute: any walk in the graph that uses edge g could instead use the edges h, i, j, k. So it's impossible to find two vertices x and y such that all $x - y$ walks rely on the existence of edge g.

In other words, edges g, h, i, j, k make a cycle, and if something happens to one of these edges, you can always "go the long way" around the cycle. In fact, this is always the reason why an edge is not a bridge!

Lemma 9.2. *In a connected graph G, edge xy is a bridge if and only if it is not on any cycles.*

Proof. Let's first formalize the idea of "going the long way around". Suppose edge xy is part of a cycle: let's say that there is a cycle

$$x_0, x_1, x_2, \ldots, x_k$$

where $x_k = x_0$, $x_i = x$, and $x_{i+1} = y$. Then there is an $x - y$ path

$$x_{i+1}, x_{i+2}, \ldots, x_k, x_1, x_2, \ldots, x_i$$

that does not use edge xy. We will use this path to show that $G - xy$ is still connected, proving that xy is not a bridge.

Let s, t be two vertices of $G - xy$. Because G is connected, G has an $s - t$ path: a sequence

$$y_0, y_1, y_2, \ldots, y_\ell$$

where $y_0 = s$ and $y_\ell = t$. If edge xy is not even used on this path then the path still exists in $G - xy$. Otherwise, if there is a place in the path where $y_j = x$ and $y_{j+1} = y$, we have the path

$$y_0, y_1, \ldots, y_{j-1}, \underbrace{x_{i+1}, x_{i+2}, \ldots, x_k, x_1, x_2, \ldots, x_i}_{\text{the } x-y \text{ path}}, y_{j+2}, \ldots, y_\ell$$

which no longer uses edge xy. (There might also be a place where $y_j = y$ and $y_{j+1} = x$; in this case, reverse the $x - y$ path before inserting it.) This shows that there is still at least an $s - t$ walk in $G - xy$; since s and t were arbitrary, $G - xy$ is still connected.

For the other direction of the proof, suppose edge xy is not a bridge. Then $G - xy$ is still connected, and in particular, $G - xy$ contains an $x - y$ path: a sequence

$$z_0, z_1, \ldots, z_m$$

with $z_0 = x$ and $z_m = y$. Then G contains a cycle using edge xy: the cycle

$$z_0, z_1, \ldots, z_m, z_0$$

whose last edge $z_m z_0$ is the edge xy. □

Question: After inserting the $x - y$ path, why don't we say that we've found an $s - t$ path in $G - xy$?

Answer: The sequence we create by insertion may no longer be a path: some vertices might now be repeated. However, it's definitely still a wlk, which is all we need.

Question: In the second half of the proof, we assumed there was an $x - y$ path in G, rather than an $x - y$ walk. Is this okay?

Answer: Yes: even though the definition of connected graphs only says there is a walk between any two vertices, we know from Theorem 3.3 that whenever an $x - y$ walk exists, an $x - y$ path also exists. (We also did this earlier with an $s - t$ path, and it was also fine there, though it was less important to know that it's a path.)

9.3 Properties of trees

What does Lemma 9.2 tell us about trees? In a tree T, every edge is a bridge, so no edge is part of any cycles. Therefore a tree T has no cycles at all.

Question: When considering multigraphs, can a tree have loops or parallel edges?

Answer: No: a loop is a cycle of length 1, and a pair of parallel edges is a cycle of length 2. So it is never necessary to model a tree as a multigraph rather than a simple graph.

This is an important property, so we'll give it a name. (In the next chapter, we'll come back and give it a second name: that's how important it is!)

Definition 9.4. *A graph is* ***acyclic*** *if it does not contain any cycles.*

Not all acyclic graphs are trees. However, the following is true:

Proposition 9.3. *A graph T is a tree if and only if it is connected and acyclic.*

Proof. Both the condition in the proposition, and the definition of a tree, say that T is connected. So we must only show that a connected graph is acyclic if and only if all its edges are bridges (the second part of the definition of a tree).

If a graph is acyclic, then it has no cycles, so none of its edges lie on cycles: by Lemma 9.2, they are all bridges. Conversely, if all edges of a graph are bridges, then by Lemma 9.2, none of them lie on cycles—so no cycles can exist at all, because a cycle needs to contain some edges. This proves the equivalence we wanted. □

Proposition 9.3 is the first in a long line of results that characterize trees, giving an if-and-only-if condition for a graph to be a tree. Every such characterization could have been the definition of trees we started with—though some are more suited to it than others. Here is one more:

Proposition 9.4. *A graph T is a tree if and only if it is acyclic, but adding any edge would create a cycle.*

Proof. Suppose T is a tree; by Proposition 9.4, we already know it is acyclic. Let e be any edge we could add to T; we want to show that $T + e$ (the graph we get if we add edge e to T) has a cycle. Well, e cannot be a bridge of $T + e$: deleting it would only give us T again, and T is connected because it is a tree. So by Lemma 9.2, e must lie on some cycle in $T + e$, and in particular, adding e to T created a cycle.

For the reverse direction, suppose that T is acyclic, but adding any edge would create a cycle. We want to prove that T is connected: then we can use Proposition 9.3 and conclude that T is a tree. To this end, let x, y be two vertices of T; we will be done if we find an $x - y$ path in T.

If xy is an edge of T, then there is such a path: a path of length 1. If not, then we know that $T + xy$ has a cycle. That cycle did not exist in T (because T is acyclic), so it must use the new edge xy. As in the proof of Lemma 9.2, when there is a cycle using edge xy, there is an $x - y$ path not using that edge that "goes the long way around" the cycle. This is an $x - y$ path that still exists in T, finishing our proof that T is connected. □

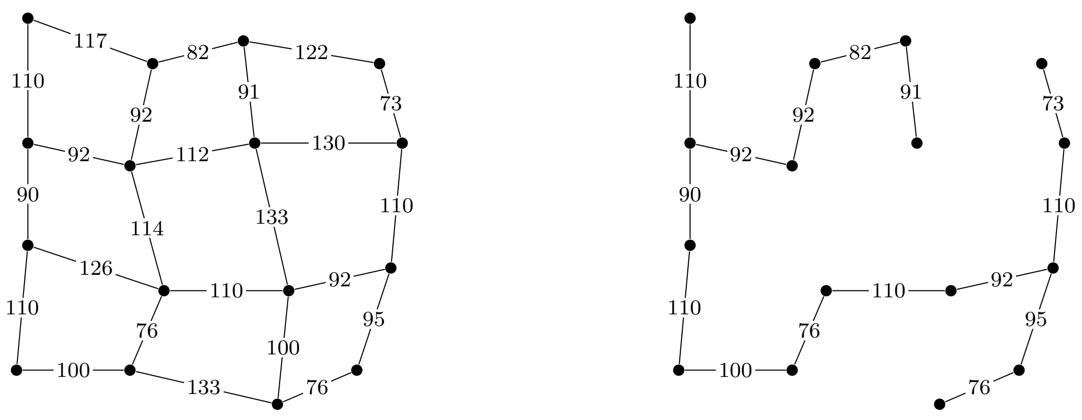
We could rephrase Proposition 9.4 to say that trees are exactly the graphs which are "maximally acyclic": acyclic with as many edges as possible, so that no more edges can be added without losing that property. In this way, it is a kind of opposite of our definition of trees as "minimally connected": connected with as few edges as possible, so that no more edges can be removed without losing that property.

9.4 Minimum-cost spanning trees

At the beginning of a chapter, we considered a transportation company which wants to find the cheapest set of routes to keep its network connected. Though we said that the transportation company wants to find a spanning tree of its network, that's not the whole story. Not all spanning trees are equally cheap, because not all of the routes in the transportation network are equally cheap to run or to maintain. To keep track of this data, we need to consider more than just a graph.

Definition 9.5. *A **weighted graph** is a graph G together with a function $w\colon E(G) \to [0, \infty)$. For an edge $e \in E(G)$, the value $w(e)$ is called the **weight** or the **cost** of e.*

Figure 9.4a shows an example of a weighted graph: what the transportation network's route map might look like. (To generate this simplified example, I placed the 16 vertices at slightly random points, and computed the distance between the points to determine the cost of the edges.)



(a) A weighted graph (b) A minimum-cost spanning tree

Figure 9.4: Finding the minimum-cost spanning tree in a weighted graph

When we measure the cost of a subgraph of a weighted graph, we compute the total cost of its edges, adding up their individual costs. A spanning tree T is a weighted graph is called a **minimum-cost spanning tree**, or **MCST** for short, if it has the minimum total cost. Figure 9.4b shows a minimum-cost spanning tree of the weighted graph in Figure 9.4a.

Question: Should the MCST just use all the cheapest edges in the graph?

Answer: No: some of the cheap edges might create a cycle with even cheaper edges, making them redundant. On the other hand, some very expensive edges might be necessary to connect the MCST.

There are many algorithms that can be used for finding the MCST of a weighted graph. Famous ones include Prim's algorithm and Kruskal's algorithm. In this chapter, I will show you the **reverse-delete algorithm** (published by Joseph Kruskal in the same paper [5] as the algorithm that bears his name), because it bears the most resemblance to our proof of Theorem 9.1.

In that proof, we repeatedly deleted edges, one at a time, until we were left with a spanning tree; however, the proof did not specify which edge to choose at each step. The only requirement is that we must never delete a bridge.

If we were to modify that strategy to find a minimum-cost spanning tree, which edges should we delete? The most natural guess is that of all the non-bridges, we should pick the most expensive: the one with the highest weight. This is a "greedy" strategy: it makes the best choice it sees in the moment, with no regard to how this affects future choices.

Question: How could a greedy strategy fail—what should we be concerned about?

Answer: It's conceivable that if we delete a cheaper edge and keep a more expensive one early on, then later in the process this will let us delete several more expensive edges to make up for it. The greedy strategy is not obviously correct: this will take some proof.

Let us summarize the reverse-delete algorithm formally. This is a slight change from the strategy we used to prove Theorem 9.1, because it only considers each edge once—however, there will be no point in returning to a previously-considered edge, because it will only be left alone if it is a bridge.

1. Let $e_1, e_2, \dots, e_m$ be a list of the edges of G in decreasing order of cost: from most expensive to cheapest. Also, let $G_0 = G$; we will construct a sequence $G_1, G_2, \dots, G_m$ of graphs as we go.
2. Starting at $i = 1$, look at edge e_i and ask: is e_i a bridge of G_{i-1}?

   If e_i is a bridge, set $G_i = G_{i-1}$; if e_i is not a bridge, set $G_i = G_{i-1} - e_i$.
3. Repeat step 2 for $i = 2, 3, \dots, m$, until all edges have been considered. Return G_m as the minimum-cost spanning tree.

We will assume that there are no ties between the costs of the edges. (One of the problems at the end of this chapter will ask you to generalize this result to allow for ties.)

Theorem 9.5. *Let G be a connected weighted graph in which all edges have distinct costs. Then the output of the reverse-delete algorithm for G will be a minimum-cost spanning tree of G.*

Proof. The algorithm always produces a connected graph, because it never deletes a bridge. Also, the only edges that are kept are bridges. Specifically, if edge e_i survives to the final graph G_m, it must first survive to G_i, which means it must have been a bridge of G_{i-1}. Therefore there is no cycle in G_{i-1} containing e_i. To obtain G_m from G_{i-1}, we only delete edges; therefore G_m also cannot have a cycle containing e_i, making e_i a bridge of G_m. Since this is true for every edge e_i, G_m must be a tree. Let's give G_m another name: call it T.

To prove that the algorithm is optimal, we need to compare T to an alternate spanning tree T', and somehow argue that T is better than T'. Specifically, what we'll do is prove that if $T' \neq T$, then T' cannot be the MCST, because we can make a small improvement to it. This will prove the theorem, because it leaves T as the only possible candidate for the MCST.

How can we do this? Well, first of all, if $T \neq T'$, we can find an edge e which is present in T, but not T'. (If every edge of T were present in T', then either T' would be equal to T, or T' would consist of T plus some additional edges. In the latter case, T' would not be a tree, because it would not be *minimally* connected.)

By Proposition 9.4, the graph $T' + e$ has a cycle. Let C be that cycle, and let e' be the most expensive edge of C.

By looking at the reverse-delete algorithm, we can prove that e' cannot be part of T, and in particular $e' \neq e$. Here's why: suppose that $e' = e_i$ in the list of edges by cost. In the graph G_{i-1} produced by the algorithm, e_i is still present, and so are all the other edges of C, because we have not gotten to any of them yet. Therefore C is a cycle of G_{i-1} containing e_i, meaning (by Lemma 9.2) that e_i is not a bridge of G_{i-1}. We conclude that e_i (that is, e') is deleted and does not survive to $T = G_m$.

Therefore we can modify T' as follows: add e, but then delete e'. This produces a cheaper connected graph!

- Why is it cheaper? Because e' is the most expensive edge of C, and e is some other edge of C: we deleted a more expensive edge than we added.
- Why is it still connected? Because we deleted an edge from a cycle of $T' + e$, which (by Lemma 9.2) was not a bridge.

Actually, it is true that $(T' + e) - e'$ is a tree itself, but proving that is inconvenient at the moment; it will be much easier once we can use Theorem 10.2 from the next chapter. However, we do not need to know that it is a tree. Since $(T' + e) - e'$ is connected, it certainly has a spanning tree by Theorem 9.1. The total cost of that spanning tree is at most the total cost of $(T' + e) - e'$, which in turn is strictly less than the cost of T'. Therefore T' cannot be the MCST.

Since this is true of any spanning tree other than T, we conclude that T is the unique spanning tree of G. □

Question: Will the tree T produced by the reverse-delete algorithm ever contain the most expensive edge, e_1?

Answer: It might, if e_1 is a bridge of G. For example, if e_1 is the only edge incident on a vertex, then we are forced to use it no matter how expensive it is.

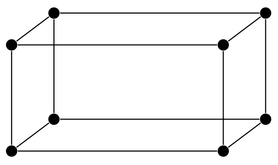
Question: Will T always contain the cheapest edge, e_m?

Answer: It will, though this is not as obvious.

To delete e_m, it would need to be part of a cycle C in G_{m-1}. However, that cycle had other edges, which we considered before e_m. Those edges were also part of C when we considered them—so why didn't we delete them?

9.5 Practice problems

1. In the diagram of the cube graph shown below, each diagonal edge has length 1, each vertical edge has length 2, and each horizontal edge has length 4.

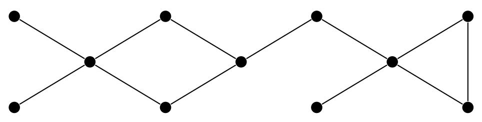


   Find a minimum-cost spanning tree of the cube graph, where the cost of each edge is taken to be its length in this diagram.

2. Recall that in the hypercube graph Q_n, vertices are n-bit strings $b_1b_2\dots b_n$, with an edge between every pair of vertices that differ in one position.

   Generalizing the previous problem, suppose we make Q_n into a weighted graph by giving an edge cost 2^k if its endpoints differ in the k^{th} bit. What will be the total cost of the minimum-cost spanning tree of this weighted graph?

3. Let G be the graph shown below:



   a) Identify all the bridges in G. *(Check: there should be* 5 *bridges.)*

   b) Find all the possible spanning trees of G. *(How does the answer to (a) help here?)*

4. Prove that an n-vertex graph with the degree sequence $n-1, 1, 1, \dots, 1, 1$ must be a tree. What does such a graph look like?

5. Find a connected 3-regular graph G which has a bridge. *(Hint: you'll need at least* 10 *vertices.)*

6. It's even harder to find a 4-regular graph which has a bridge.

   a) Let G be a 4-regular graph in which edge xy is a bridge. Then $G-xy$ should have two components: one containing x, and one containing y. Describe the degree sequences of each component.

   b) Conclude that this can't happen: a 4-regular graph cannot have a bridge.

7. Prove that T is a tree if and only if, between any two vertices of G, there is *exactly one* path. (This is another of the many characterizations of trees to accompany Proposition 9.3 and Proposition 9.4.)

8. Theorem 9.5 assumes that all edges of the graph have distinct costs: there are no ties.

   a) Prove that if there *are* ties between the costs of some edges, then we can still conclude one of two things in the proof: either we find a spanning tree cheaper than T' (so T' cannot be the MCST) or $(T' + e) - e'$ is a spanning tree with the same cost as T', but "closer" to T in some sense.

   b) Explain why this is enough to still deduce that T is an MCST (even if it might not be unique).

   c) Use this to prove that every spanning tree of the cube graph Q_3 must have 7 edges. (Do not, of course, use my claim that every spanning tree of Q_3 is isomorphic to one of the six trees in Figure 9.2.)

10 Properties of trees

The purpose of this chapter

Mathematically, the new properties of trees you will learn in this chapter are nothing special: in Theorem 10.1 and Theorem 10.2, we will prove how many edges they have, and later in Lemma 10.5 and Lemma 10.6, we will learn about their degree-1 vertices.

All this is notable because it is the point when trees suddenly begin pulling their weight as theoretical tools that help us solve problems. I have included two examples of this: the flower garden problem (based on my experience with recreational mathematics) and Corollary 10.9 (based on my experience with mathematical olympiads). In both cases, we are not solving a problem about trees: we are solving a problem that trees help us solve, because the problem is somehow related to a connected graph.

Lemma 10.6 is also important because it is what makes trees such an excellent setting for proofs by induction.

10.1 A square garden

Suppose you are designing a garden in the shape of a square $n \times n$ grid. Each cell of the grid can either contain flowers, or be part of a garden path for visitors. The garden path can bend and fork however you like, but it must be connected: visitors to the garden must be able to see all of it! Also, every cell of the grid with flowers must be next to a cell of the garden path—otherwise, visitors will not be able to see the flowers, and gardeners will not be able to water them. In both cases, only horizontal and vertical adjacencies count, not horizontal ones.

What is the largest number of cells of the grid that can contain flowers?

I have drawn some solutions for 4×4, 5×5, and 6×6 flower gardens in Figure 10.1, to show you what they can look like. If you'd like to try it yourself, see if you can manage to fill 29 cells of a 7×7 garden with flowers.

Solving the problem exactly for large $n \times n$ grids is, as far as I know, very difficult even with the help of a computer. However, with the aid of the material in today's chapter, we will able to establish a very good upper bound on the number of flowers!

To do so, we will need to recast the problem as a graph. There are two relevant graphs at work here, actually. The first is the grid graph, defined below:

Definition 10.1. *For any $m \geq 1$ and $n \geq 1$, the* ***$m \times n$ grid graph $G(m, n)$*** *is the graph with mn vertices: the points (x, y) where $x \in \{1, \dots, m\}$ and $y \in \{1, \dots, n\}$. Two vertices are adjacent precisely when the points are at distance 1 from each other (in the Cartesian plane).*

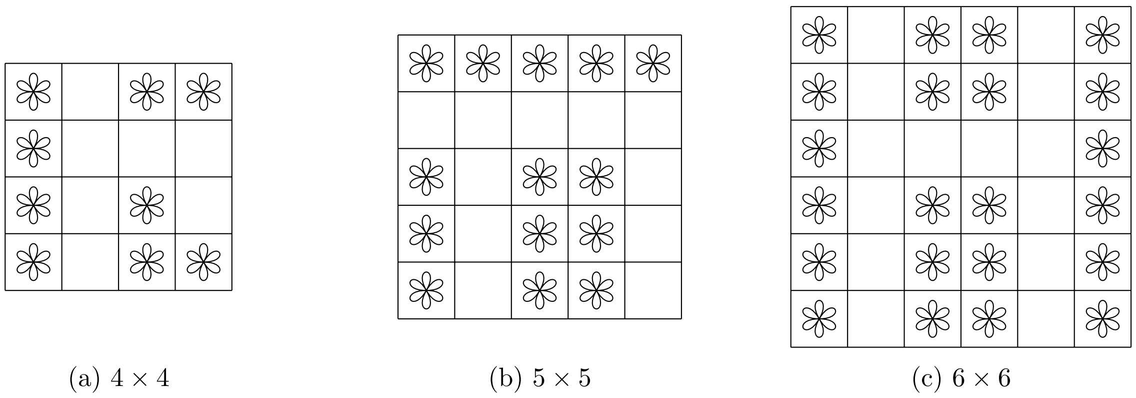

(a) 4×4 (b) 5×5 (c) 6×6

Figure 10.1: Some high-density flower gardens

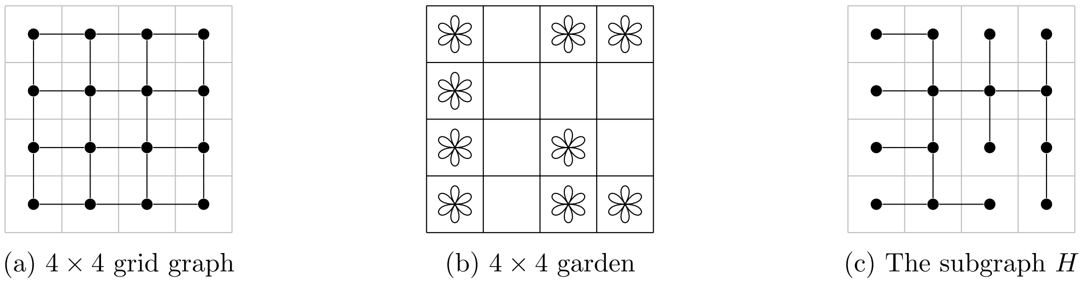

(a) 4×4 grid graph (b) 4×4 garden (c) The subgraph H

Figure 10.2: Representing a flower garden as a graph

The $n \times n$ grid graph $G(n, n)$ represents the $n \times n$ flower garden: its vertices correspond to the cells of the grid that forms the garden, and its edges correspond to adjacent cells. The $n = 4$ case of this is illustrated in Figure 10.2a. Of course, $G(n, n)$ does not tell us anything about the solution; it is merely the setting where the garden is posed. For simplicity, we will forget about the grid in favor of the grid graph: we will say that we plant flowers on some of the vertices of $G(n, n)$, and put a garden path on the rest.

We can think of a solution to the flower garden problem as a spanning subgraph H of $G(n, n)$. We include only the edges of $G(n, n)$ relevant to checking the solution. First, keep all edges of $G(n, n)$ between two vertices that are both part of of the garden path: these will be necessary to check that the garden path is connected. Second, for every vertex where a flower is planted, keep *one* edge to an adjacent garden path vertex: these are necessary to check that the flower is accessible from the garden path. Figure 10.2b and Figure 10.2c show how a 4×4 solution turns into a subgraph H.

Every valid flower garden turns into a subgraph H, but which subgraphs are the best? They are the subgraphs with the most degree-1 vertices, because those are the vertices where flowers are planted. (In principle, a garden path vertex can also end up having degree 1, if it is an end of the garden path and not necessary to visit any flowers; however, in that case, we would do better to just plant a flower there, instead.)

In Chapter 4, we called a vertex with degree 1 a **leaf**. So we are looking for a connected spanning subgraph of H with as many leaf vertices as possible. Though there is no requirement for H to be a spanning tree, you may notice that the graph in Figure 10.2c is in fact a tree. This is no coincidence: cycles in H can only appear in the garden path, but even there they are wasted adjacencies we'd rather use to plant more flowers.

How do we maximize the number of leaves in the spanning tree? To do this, we need to combine some knowledge of vertex degrees with some facts about the number of edges in a tree, which we will prove later in this chapter. I will postpone discussing the graph theory behind the flower garden problem any further until you've seen that necessary background.

Before we move on to more theoretical questions, let me say a bit about the background of this problem. I made up the flower-garden formulation of it for this book, but I've seen many questions like it before. Mostly these have not been asked by professional mathematicians, but by people analyzing board games and video games with a square grid. However, while writing this chapter, I looked up and found a paper by Li and Toulouse [6] studying this exact problem: "Maximum Leaf Spanning Tree Problem for Grid Graphs" by Li and Toulouse. The paper confirmed that my solutions for the 4×4 and 6×6 grid are optimal, and informed me that solving it is useful for practical questions in the areas of networking and circuit layout!

10.2 Counting edges in trees

By Proposition 9.3 from the previous chapter, a tree is an acyclic, connected graph. What can we say about the number of edges this requires?

In Chapter 4, we looked at the relationship between the number of edges in a graph, and cycles in the graph. Specifically, Corollary 4.6 tells us that an n-vertex graph with at least n edges is guaranteed to contain a cycle.

Question: What does this tell us about the number of edges in an n-vertex tree?

Answer: To avoid creating any cycles, the tree can have at most $n - 1$ edges.

Question: If an n-vertex graph has $n - 1$ edges, must it be a tree?

Answer: No: there's no reason to conclude the graph is either acyclic or connected. For example, we could start with a cycle that has $n-1$ vertices and edges, then add an isolated vertex: this is a graph with n vertices and $n - 1$ edges which is not a tree.

If this were all we knew about trees, then we'd have to stop there, because n-vertex graphs without cycles can have any number of edges between 0 and $n-1$: there's nothing forcing them to have any edges. What forces trees to have edges, on the other hand, is the requirement to be connected: we need some edges to connect the tree! Let's explore how many.

Theorem 10.1. *A graph with n vertices and m edges has at least $n-m$ connected components.*

Proof. We'll prove this theorem by induction on m: just as in our proof of the Handshake Lemma in Chapter 4, we will see what happens as we add edges to a graph one at a time. This time, however, what we'll be paying attention to is the number of connected components.

Our base case is $m = 0$. If a graph has n vertices and 0 edges, then every vertex is an isolated vertex, so it is a connected component all by itself. There are always exactly $n = n - m$ connected components.

Now assume that the theorem is true for graphs with $m-1$ edges, and let G be a graph with n vertices and m edges. Let xy be an arbitrary edge of G, and consider the $(m-1)$-edge graph $G - xy$. By the induction hypothesis, $G - xy$ has at least $n - m + 1$ connected components.

There are two cases for what edge xy can do to change this:

- If x and y are in the same connected component of $G - xy$, then adding edge xy does not do anything at all. There is already an $x - y$ path, so any walk that used edge xy could have used that $x - y$ path instead. Therefore if two vertices are in the same component of G, they're also in the same component of $G - xy$; G also has at least $n - m + 1$ connected components.

- If x and y are in different connected components of $G - xy$, then those two components become the same connected component of G. Any vertex in x's component can reach y (by going to x, then taking edge xy) and from there, it can reach any vertex in y's component.

  However, that's all that happens. If we look at a vertex z not in either of these connected components, then there's no walk starting at z that uses edge xy: it would first have to get to x or y without using that edge, which is impossible. So walks from z are the same in G and in $G - xy$.

  As a result, G has one connected component less than $G - xy$: at least $n - m$ connected components.

In both cases, G has at least $n - m$ connected components, so the induction is complete and the statement we want is true for all n and m. □

We can use these ideas to prove a few more characterizations of trees. Let's do that, and summarize all our results so far in one big theorem:

Theorem 10.2. *The following conditions for a graph G with n vertices are all equivalent definitions of a tree:*

1. *G is minimally connected: connected, but deleting any edge will disconnect it.*
2. *G is maximally acyclic: acyclic, but adding any edge will create a cycle.*
3. *G is both acyclic and connected.*
4. *G is connected and has at most $n - 1$ edges.*
5. *G is acyclic and has at least $n - 1$ edges.*
6. *G is uniquely connected: there is exactly one path between any two vertices.*

Proof. We already know that conditions 1–3 are equivalent; we proved that last time. Assuming condition 3, we can:

- Use Corollary 4.6 to conclude that G can have at most $n-1$ edges to be acyclic, proving condition 4;
- Use Theorem 10.1 to conclude that G must have at least $n-1$ edges to be connected, proving condition 5.

So conditions 4–5 are true of all trees.

Suppose that condition 4 holds. Then G is not only connected but minimally connected: if we delete any edge, then the resulting $n-2$ edges are not enough to connect G, by Theorem 10.1. Therefore condition 1 holds, and G is a tree.

Suppose that condition 5 holds. Then G is not only acyclic but maximally acyclic: if we add any edge, then the resulting n edges are guaranteed to contain a cycle, by Corollary 4.6. Therefore condition 2 holds, and G is a tree.

This shows that conditions 1–5 are equivalent. I've included condition 6 from a practice problem at the end of Chapter 9; it's also equivalent to the rest, but I won't prove that here. □

Question: In condition 4 of Theorem 10.2, why do we say that G "has at most $n-1$ edges", when in fact we can conclude G has exactly $n-1$ edges?

Answer: One of the reasons to have many characterizations of a tree is to make it as easy as possible to prove that a graph G is a tree. So conditions 4 and 5 are given weaker statements to make them easier to prove: if G is connected, and we can easily check that it cannot have more than $n-1$ edges, we don't need to check that it has exactly that many edges.

10.3 From trees to forests

The statement of Theorem 10.1 is an inequality: the number of connected components is *at least* $n-m$. In mathematics, every inequality we prove is also a challenge to understand when the inequality is **tight**: when the $\le$ turns into $=$. For example, the inequality in Theorem 10.1 is tight when the graph is a tree: a tree with n vertices has $n-1$ edges and exactly $n-(n-1)=1$ connected components. But are there other examples?

If k is the number of connected components, then the inequality $k \ge n-m$ can be rephrased as $m \ge n-k$: a graph with n vertices and k connected components must have at least $n-k$ edges. To get an example where this inequality is tight, we want to use as few edges as possible.

Question: If we want to reach the minimum number of edges, what should we do in each connected component?

Answer: We want each connected component to be a tree, because this uses the fewest edges to keep the component connected.

So we define:

Definition 10.2. *A graph F is a **forest** if each connected component of F is a tree.*

Question: In fact, we recently defined a different term for graphs that are forests. What is it?

Answer: Forests are exactly the same as acyclic graphs. A tree is acyclic and connected: in a forest, each connected component is a tree, which means there are no cycles, but we've abandoned the requirement of being connected.

Forests are also exactly the graphs for which the inequality of Theorem 10.1 is tight. (This statements includes trees as a special case: in graph theory, a single tree is still a forest!)

Proposition 10.3. *A graph with n vertices and m edges has exactly $n-m$ connected components if and only if it is a forest.*

Proof. Suppose an n-vertex forest has k connected components: trees with $n_1, n_2, \dots, n_k$ vertices, where $n_1 + n_2 + \cdots + n_k = n$. The i^{th} tree has $n_i - 1$ edges, so the total number of edges in the forest is

$$(n_1 - 1) + (n_2 - 1) + \cdots + (n_k - 1) = (n_1 + n_2 + \cdots + n_k) - k = n - k.$$

Since $m = n - k$, we have $k = n - m$, proving one direction of the proposition.

To prove the other direction, suppose for the sake of contradiction that G is a graph with n vertices, m edges, and exactly $n - m$ connected components which is *not* a forest. Then G contains a cycle; let e be any edge on this cycle.

By Lemma 9.2, e is not a bridge of its connected component: in $G-e$, that connected component will remain connected. This means $G - e$ still has exactly $n - m$ connected components. But $G - e$ has n vertices and $m - 1$, so by Theorem 10.1, $G - e$ must have $n - m + 1$ connected components: a contradiction! □

10.4 Leaves in trees

To solve the flower garden problem, we were interested in the number of leaves—degree 1 vertices—of a spanning tree of the $n \times n$ grid graph. Now that we know the number of edges in a tree, the Handshake Lemma is sufficient to get a fairly precise upper bound. (I am going to state the lemma in a more general form, to allow for the possibility of cycles in the garden path and unused squares in the garden, but in spirit it is about spanning trees.)

Proposition 10.4. *A connected subgraph of the $n \times n$ grid graph $G(n, n)$ can contain at most $\frac{2}{3}(n^2 + 1)$ leaves, and therefore an $n \times n$ flower garden can contain at most $\frac{2}{3}(n^2 + 1)$ flowers.*

Proof. Let H be a connected subgraph of $G(n,n)$. We know that H has k vertices for some $k \le n^2$, and by Theorem 10.1, we know that H has at least $k-1$ edges.

Suppose that H has l leaves. The remaining $k-l$ vertices of H can have degree at most 4, because the vertices of $G(n,n)$ have degree at most 4. Therefore if we add up the degrees of all k vertices of H, we get a total of at most

$$\underbrace{1+1+\cdots+1}_{l \text{ times}}+\underbrace{4+4+\cdots+4}_{k-l \text{ times}} = l+4(k-l) = 4k-3l.$$

By the Handshake Lemma, this total must be equal to twice the number of edges in H, giving us the inequality

$$4k-3l = 2|E(H)| \ge 2(k-1).$$

Solving for l, we get $3l \le 2k+2$, or $l \le \frac{2}{3}(k+1)$. Since $k \le n^2$, we obtain the bound we wanted: $l \le \frac{2}{3}(n^2+1)$. □

The inequality in Proposition 10.4 is not quite tight: it is impossible to reach exactly $\frac{2}{3}(n^2+1)$ flowers. For example, the garden in Figure 10.1c has 22 flowers, while $\frac{2}{3}(6^2+1) = 24\frac{2}{3}$. (I have written the answer as a mixed integer, which is unusual in advanced math, to make it clear that the bound rounds down to 24.)

Part of the gap is due to number theory: actually, $\frac{2}{3}(n^2+1)$ is never an integer for any n, so the sharper bound $\frac{2}{3}n^2$ is also true. Part of the gap is that we can never obtain the ideal scenario where all vertices of H have degree 1 or 4. However, the true answer to the problem is always very close to $\frac{2}{3}n^2$, as I will ask you to prove in one of the practice problems at the end of this chapter.

In this application, we wanted to achieve the maximum number of leaves possible; it is also often useful to know what the minimum number of leaves is.

Lemma 10.5. *Every tree with at least* 2 *vertices has at least two leaves.*

Proof. Consider a tree with n vertices, l of which are leaves. When $n \ge 2$, it is impossible for the tree to contain degree-0 vertices: such a vertex is an isolated vertex, and cannot be part of a larger connected graph. Therefore the remaining $n-l$ vertices have degree at least 2.

We can now apply the Handshake Lemma, as we did in the proof of Proposition 10.4. We know that the sum of all n degrees in the tree is $2n-2$: twice the number of edges. However, we also know that the sum is at least

$$\underbrace{1+1+\cdots+1}_{l \text{ times}}+\underbrace{2+2+\cdots+2}_{n-l \text{ times}} = l+2(n-l) = 2n-l.$$

From the inequality $2n-l \le 2n-2$, we naturally deduce $l \ge 2$: the tree must have at least two leaves. □

Question: Why do we have a $\le$ inequality ($2n-l \le 2n-2$) in this proof, when we got a $\ge$ inequality ($4k-3l \ge 2k-2$) in the proof of Proposition 10.4?

Answer: In this proof, 2 is a lower bound on the degree of the non-leaf vertices; in the previous proof, 4 was an upper bound on the degree of the non-leaf vertices.

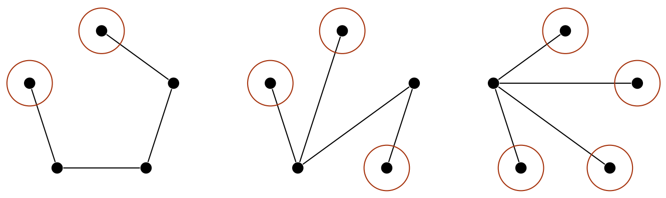


Figure 10.3: Leaves in the trees with 5 vertices

Figure 10.3 shows all three possible 5-vertex trees, up to isomorphism, with the leaves circled. You can see that the number of leaves can vary; in this case, it varies from 2 (on the left) to 4 (on the right).

Question: Is it always possible to have an n-vertex tree with exactly 2 leaves?

Answer: Yes: the path graph P_n is a tree, because it is connected and has $n-1$ edges, and only the start and end of the path are leaves.

Question: For an n-vertex tree, what is the maximum number of leaves?

Answer: It is $n-1$ (at least when $n>2$): imitating the last tree in Figure 10.3, the graph with a single central vertex adjacent to $n-1$ leaves is an n-vertex tree. This graph is called a **star graph**.

When $n>2$, all n vertices cannot be leaves, because then the degree sum would be n, which is less than $2n-2$. However, this *is* achievable when $n=2$: the only 2-vertex tree, P_2, has 2 leaves.

10.5 Induction on trees

Why is it important to have leaves? Because when a tree has leaves, we can pluck them to make the tree a tiny bit smaller.

Lemma 10.6. *If T is a tree and x is a leaf of T, then $T-x$ is also a tree.*

Proof. We have an abundance of conditions in Theorem 10.2 that we can check to prove this lemma. I think the easiest one to use is condition 5.

$T-x$ is acyclic because it's a subgraph of T, which itself has no cycles. If T has n vertices and $n-1$ edges, then $T-x$ has $n-1$ vertices and $n-2$ edges (because deleting x also deletes the single edge incident on x). The number of edges is one less than the number of vertices, so condition 5 of Theorem 10.2 is satisfied: $T-x$ is a tree. □

Lemma 10.6 is useful in a proof by induction. In Appendix B, I explain why it is important to write your induction proofs backward: having assumed the induction hypothesis for $(n-1)$-vertex graphs, we must consider an n-vertex graph and remove a vertex from it to apply the induction hypothesis. Lemma 10.6 gives us a convenient vertex to remove: if we are proving a

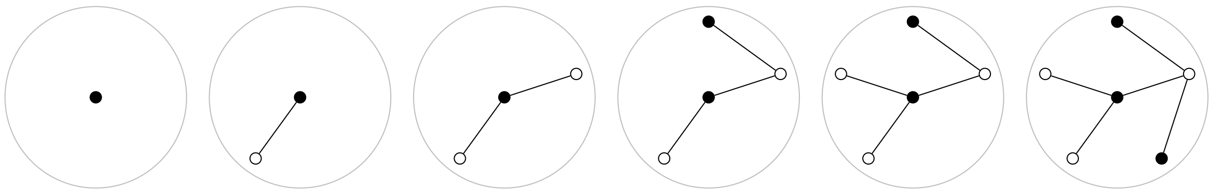


Figure 10.4: Coloring a tree using the proof of Proposition 10.7

theorem by induction about trees, then we can remove a leaf from an n-vertex tree to get an $(n-1)$-vertex tree.

The following example will, eventually, turn out to be a silly one. It is concerned with **graph coloring**, which we will consider in general in Chapter 16. However, coloring with only 2 colors is a special case: a graph that can be 2-colored is a **bipartite graph**, and in Chapter 12, we will prove a general theorem that makes the upcoming Proposition 10.7 a corollary with a one-line proof.

For now, though, it is a good way to practice induction on trees—and later, the ability to prove this result much more easily will give you a well-deserved feeling of power.

Proposition 10.7. *The vertices of every tree can be colored black and white such that no two vertices of the same color are adjacent.*

Proof. We induct on n, the number of vertices in the tree. When $n = 1$, there is only one vertex, so we may give it any color we like without violating the condition.

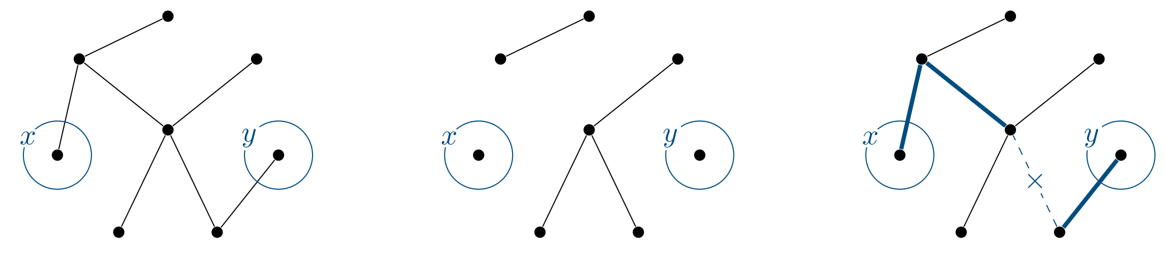
Now assume, for some $n \geq 2$, that every $(n-1)$-vertex tree has a black-and-white coloring in which no two vertices of the same color are adjacent. (Such a black-and-white coloring is called a **proper 2-coloring**.) Let T be an arbitrary n-vertex tree; we will show that T also has a proper 2-coloring.

Let x be a leaf of T (which exists by Lemma 10.5), and let y be its only neighbor in T. By Lemma 10.6, $T - x$ is also a tree; it has $n - 1$ vertices, so by the induction hypothesis, it has a proper 2-coloring.

To color T, first give every vertex the same color that it had in $T - x$. Then, color x by the following rule: if y is white, color x black, and if y is black, color x white. This rule ensures that x and y do not have the same color; the same is true for every other pair of adjacent vertices, because they were already adjacent in $T - x$, and $T - x$ was given a proper 2-coloring.

This shows that T also has a proper 2-coloring, completing the induction step. By induction, trees with any number of vertices have proper 2-colorings, completing the proof. $\square$

The proof of Proposition 10.7 is not just a proof: like many proofs by induction, it contains a recursive algorithm. Given an n-vertex tree, we can color it by choosing a leaf, removing it, and applying the coloring algorithm to the $(n-1)$-vertex tree that remains, before putting back the leaf we removed and coloring it as the n^{th} vertex. Although the recursive algorithm proceeds backwards from an n-vertex tree to a 1-vertex tree, if you think about the order in which vertices are colored, it will appear as though we started from 1 vertex and grew the tree



(a) T, with x and y circled (b) The subgraph H (c) The final result, H'

Figure 10.5: The induction step of Proposition 10.8

by adding a succession of leaves, coloring them as we go. Figure 10.4 shows an example of coloring a 6-vertex tree using this strategy.

Question:	Can you deduce from the proof *how many* proper 2-colorings a tree has?
Answer:	In the induction step of the proof, the color of x was forced: it had to be the opposite color of y. Similarly, at every previous step, the color of the leaf is forced. However, in the base case, we could give the single vertex of a 1-vertex tree either color. So there are 2 proper 2-colorings possible, based on which color we chose at that step. (They are opposites of each other: one is obtained from the other by switching black and white.)

In most proofs by induction, the full power of Lemma 10.5 is not necessary: a single leaf is enough. Here is an example where we really must use the existence of two leaves.

Proposition 10.8. *Every tree with an even number of vertices has a spanning subgraph (not necessarily connected) in which every vertex has odd degree.*

Proof. As before, we induct on n, the number of vertices in the tree. Because the proposition only considers trees with an even number of vertices, our base case is $n = 2$. In a tree with 2 vertices, both vertices are leaves, so the tree itself is the spanning subgraph we need.

Now assume, for some even $n \geq 4$, that the proposition is true for all $(n-2)$-vertex trees. (We go back from n to $n-2$, the previous even number.) Let T be an arbitrary n-vertex tree; by Lemma 10.5, T has two leaves, which we call x and y. Figure 10.5a shows an example of such a tree, in which you can follow along as we carry out the induction step.

Applying Lemma 10.6 twice, the subtree $(T-x)-y$ is an $(n-2)$-vertex tree. So it has a spanning subgraph in which every vertex has odd degree. Add x and y back into this subgraph (as isolated vertices) and call the result H. (The graph H in our example is shown in Figure 10.5b.)

This H is a spanning subgraph of T, but what about the degrees? Vertices x and y have degree 0 in H, by construction, which is even. However, every other vertex has an odd degree in H, just as we wanted. All that's necessary is to fix x and y.

At first, this seems impossible. Suppose you add the edge incident on x to H. Then vertex x now has odd degree, but its neighbor has even degree. If you make a change to fix that neighbor, another vertex will go from odd to even, and so on.

This is the reason that we need to work with two leaves at once. In T, there is an $x - y$ path. Change H to a new graph H' by toggling every edge along the $x - y$ path in T. (That is, for every edge of the path, if it is not in H, add it, and if it is in H, remove it. If the explanation is not clear, see Figure 10.5c for an example.)

We can check by cases that every degree in H' is odd:

- A vertex not on the $x - y$ path is untouched, and still has odd degree.
- A vertex in the middle of the $x - y$ path either gained two edges, gained an edge and lost an edge, or lost two edges. The change in degree is $+2$, $+0$, or -2, so the degree remains odd.
- Finally, x and y gain an edge (because the edges incident on x and on y were not in H before), so their degree goes from 0 to 1: an odd number.

Therefore T has a spanning subgraph in which every vertex has odd degree: the subgraph H'. By induction, the same is true for every tree with an even number of vertices. □

Question:	Does any tree with an odd number of vertices have a spanning subgraph such as the one in Proposition 10.8?
Answer:	No: that would be a graph with an odd number of vertices, all with odd degree, which cannot exist by the Handshake Lemma.

This example is also an illustration of an idea that I first mentioned in Chapter 9: if we must prove a theorem for connected graphs, and the theorem is only helped by having more edges, then it's enough to prove the theorem for trees—which will be the hardest case.

Here, once we have Proposition 10.8, we may immediately obtain the following corollary, which was proven by Atsushi Amahashi in 1985 [4], and later appeared as a problem on the 2005 Bay Area Math Olympiad [1].

Corollary 10.9. *Every connected graph with an even number of vertices has a spanning subgraph (not necessarily connected) in which every vertex has odd degree.*

Proof. Apply Proposition 10.8 to a spanning tree of the graph. □

10.6 Practice problems

1. Let T be tree whose degree sequence has the form $4, 3, 2, 1, 1, 1, \dots$ (that is, $4, 3, 2$ followed by some number of 1's).
   a) Determine the number of 1's in the degree sequence of T.
   b) There is more than one possibility for a tree T with this degree sequence. Give two non-isomorphic trees with this degree sequence, and explain why they are not isomorphic.
2. Find all 6-vertex trees up to isomorphism. (There are six of them.)

3. Let G be a graph with 10 vertices and 10 edges.

   a) If G contains exactly one cycle, how many connected components must it have? Give an example of such a graph.

   b) If G contains exactly two cycles, how many connected components must it have? Give an example of such a graph.

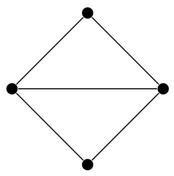
   c) If G contains exactly three cycles, there's two possible values for the number of connected components. Why is that? Give examples for each possibility.

   (In this problem, we consider two cycles to be the same if they are the same as subgraphs of G, not if they are the same as sequence of vertices. For example, the cycle x, y, z, x is the same cycle as z, y, x, z.)

4. Prove that when n is a multiple of 3, there is a solution to the $n \times n$ flower garden problem which contains at least $\frac{2}{3}(n^2 - n) + 2$ flowers.

   (You will have to come up with a generalizable layout for the flower garden which contains this many flowers—see if you can generalize the layout in Figure 10.1c.)

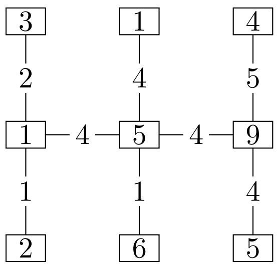
5. Find all 8 spanning trees of the graph below. (I do not care about isomorphism in this problem; if several spanning trees are isomorphic but different graphs, find them all.)



6. a) Prove that if x and y are vertices of a graph G in the same connected component, then adding edge xy to G creates at least one new cycle.

   b) Imitate the proof of Theorem 10.1 to prove that a graph with n vertices and m edges has at least $m - n + 1$ cycles.

7. a) Let T be a tree in which every edge e is labeled by a positive integer. Prove that it's possible to place positive integer labels on the vertices of T such that for any two adjacent vertices x and y, the absolute difference between their labels is exactly the label of edge xy. An example is shown in the diagram below:



   (Of course you can go from the vertex labels to the edge labels; that's just subtraction. I am asking you to go from the edge labels to the vertex labels.)

   b) Find an example of a graph (not a tree!) in which every edge is labeled by a positive integer in such a way as to make the task in part (a) impossible.

8. Prove that if T is a k-vertex tree and G is a graph with minimum degree at least $k - 1$, then G contains a subgraph isomorphic to T.

9. Prove that for any sequence of $n \geq 2$ positive integers whose sum is $2(n - 1)$, there is a tree with that degree sequence.

   (Compare this to the complicated procedure we developed in Chapter 4 to determine if there is any graph at all with a given degree sequence!)

11 Cayley's formula

The purpose of this chapter

I am including a chapter on Cayley's formula for the number of labeled trees with n vertices in this book, first of all, because it is a beautiful piece of mathematics. It comes with an introductory look at counting problems in graph theory, which I think is especially important to include because I often tend to overlook such problems when I think about what a graph theorist needs to know. (The next chapter is also about counting problems in graph theory, but from a very different perspective.) In short, even though Cayley's formula is not necessary as a prerequisite for any of the following chapters, I think it is a worthwhile topic to study on its own.

I do not like including sections such as the last section of this chapter, which is mainly about results too complicated to prove in this book. However, I think that a textbook must say something about the unlabeled counting problem even if it simply says that not very much is known about the problem. When teaching this topic, I would not have feelings that are nearly as strong; I think it is perfectly acceptable to say that the textbook has a page on the unlabeled version of the problem, and leave it at that.

11.1 How to count graphs

This chapter is the first in which we will seriously look at counting problems in graph theory. These are usually questions of the form "How many graphs satisfying such-and-such property are there?" or "Given a graph G, how many objects of such-and-such type does G contain?"

When we count graphs, we must first choose between two options that are typically described as "counting labeled graphs" and "counting unlabeled graphs". This is a bit misleading, because all graphs are labeled, in the sense that all graphs have a vertex set with distinguishable elements, but it does convey the right intuition. Here is how to describe these options more precisely.

Counting labeled graphs is typically the easier of the two problems. To describe it more precisely, suppose that we being by choosing a set V of size n: it does not matter too much which set V, as long as we choose it, so choosing $V = \{1, 2, \dots, n\}$ is a very reasonable and simple choice. Then we ask: how many graphs with vertex set V have a certain property?

Counting unlabeled graphs is typically harder, because questions of symmetry and group theory often enter the picture. Although you might imagine that a "labeled graph" is a graph that used to be unlabeled and then received labels, mathematically the relationship between the problems is the opposite. To count unlabeled graphs of some type, we start with the set $\mathcal{S}$ of all labeled graphs of that type (again, with some fixed vertex set V). Commonly, many of the graphs in $\mathcal{S}$ will be isomorphic. Graph isomorphism is an equivalence relation on $\mathcal{S}$, so it partitions $\mathcal{S}$

into equivalence classes. The unlabeled problem, formally, asks: how many equivalence classes are there? In other words, if we consider isomorphic graphs in $\mathcal{S}$ to be the same, how many different ones are there?

In this chapter, we will count n-vertex trees. Most of the chapter is devoted to the labeled counting problem, for which a clean and beautiful formula exists. Much less is known about the unlabeled counting problem, but I will summarize what is known about it at the end of the chapter.

Now that we've made a decision about which problem we're counting, we need some counting techniques. Studying these is a whole branch of combinatorics, so naturally I cannot explain all counting techniques for you in half a page. I will just give you one particular approach to counting problems: we can count elements a set by first finding an encoding for these elements, and then counting the number of valid encodings. That's just replacing one counting problem by another, of course—but if we choose the right encoding, then it will be an easier problem. Here are two examples.

Problem 11.1. *How many labeled graphs on n vertices are there?*

Answer to Problem 11.1. We will encode these graphs as binary strings of length $\binom{n}{2}$. Here, $\binom{n}{2} = \frac{n(n-1)}{2}$ is the number of edges in the complete graph K_n, which we will also consider to have vertex set $\{1, 2, \dots, n\}$. To encode a graph G with $V(G) = \{1, 2, \dots, n\}$, we go through the edges of K_n in a fixed order, such as the dictionary order

$$12, 13, \dots, 1n, 23, 24, \dots, 2n, \dots.$$

For each edge, we ask: does G also contain that edge? If so, we write down a 1; if not, we write down a 0.

At the end, the sequence of $\binom{n}{2}$ zeroes and ones **uniquely encodes** G: we can recover G from the sequence, and there is only one sequence from which we recover G and not any other graph. Moreover, all sequences of $\binom{n}{2}$ zeroes and ones are **valid encodings**: they correspond to one of the graphs we want to count. Together, these two claims tell us that there is a bijection between the graphs we're counting and their encodings—so we can count the encodings instead of counting the graphs.

There are $2^{\binom{n}{2}}$ sequences of $\binom{n}{2}$ zeroes and ones (because we have 2 choices for each bit in the sequence) and therefore there are $2^{\binom{n}{2}}$ labeled graphs on n vertices. □

Problem 11.2. *How many labeled 1-regular graphs on n vertices are there?*

Answer to Problem 11.2. As we know from Chapter 5, such graphs only exist when n is even. In the cases of even n, we could begin by listing all $n/2$ edges in the graph. This is done in Figure 11.1 in the case $n = 4$.

Question:	Are there multiple ways to list the edges in a graph in a sequence?
Answer:	Yes: for each edge xy, we can also write it as yx, and we can also write the edges in any order.

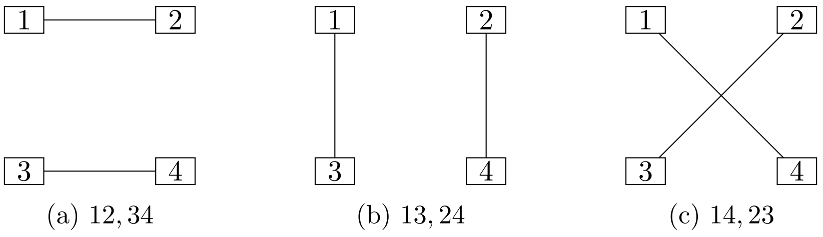


Figure 11.1: The 1-regular graphs with vertex set $\{1, 2, 3, 4\}$

To make the encoding a unique encoding, we should make a rule for how the edges should be written, and in which order they should appear. A simple option is to write a "sorted sequence of sorted edges": to sort each edge xy so that $x < y$, and to sort the sequence of edges by the first number in each pair. This is already done in Figure 11.1.

Question: Are all "sorted sequences of $n/2$ sorted edges" valid encodings?

Answer: No: some of the encode graphs with $n/2$ edges that are not 1-regular, by leaving out some vertices and using others too many times. Each vertex must appear exactly once.

If not all sequences are valid encodings, this complicates our life, but we can still proceed. We just have to describe which encodings are valid, and count only the valid ones.

If the encoding scheme we've chosen is a good one, we will be able to count valid encodings by the **product principle**: going from left to right, there will be a fixed number of options for the number in each position, and by multiplying together the number of choices at each step, we can count the overall number of encodings.

Question: Going from left to right, what are the constraints on the numbers in the "sorted sequence of sorted edges"?

Answer: The first number in each edge is uniquely determined: it is simply the smallest element of $\{1, 2, \dots, n\}$ that has not yet appeared. The second number in each edge can be *any* of the numbers that have not yet appeared.

Question: Going from left to right, how many options are there for each number?

Answer: There is 1 option for the first number in each edge, and $n - 2k + 1$ options for the second number in the k^{th} edge.

Multiplying together these options, we get a product

$$\underbrace{1 \cdot (2n-1) \cdot 1 \cdot (2n-3) \cdot 1 \cdot (2n-5) \cdots 1 \cdot 3 \cdot 1 \cdot 1}_{n \text{ factors}}$$

in answer to the problem. We can ignore the factors of 1 and just say that this is the product of the first $n/2$ odd positive integers. For example, we get $3 \cdot 1 = 3$ when $n = 4$ (as seen in

Figure 11.1), $5 \cdot 3 \cdot 1 = 15$ when $n = 6$, $7 \cdot 5 \cdot 3 \cdot 1 = 105$ when $n = 8$, and so on. This product is often written with the **double factorial** symbol $(2n-1)!!$. □

11.2 Trees and deletion sequences

Having solved a few problems for practice, we can move on to the question we're really interested in: how many labeled trees on n vertices are there?

The solution to Problem 11.2 gives us a good starting point. An n-vertex tree, as we know, is in particular an $(n-1)$-vertex graph. So we can begin by writing down the edges of that graph, in a convenient order.

The most convenient order to use here is not the dictionary order we used before. We would like to make use of the structure of a tree! In particular, we know from Lemma 10.6 in the previous chapter that if we remove a leaf vertex (and its only edge) from a tree, we get a smaller tree. We used this to great effect to write inductive proofs of theorems about trees; it can be used to equally great effect to write recursive algorithms for problems about trees.

Given a tree T with vertex set $\{1, 2, \dots, n\}$, here is an algorithm to list all its edges in a uniquely defined order:

1. If $n = 1$, then there are no edges, so the list of edges should be empty as well.
2. Otherwise, for $n > 1$, the tree T will have some leaves. Let x be the lowest numbered leaf, and let y be its neighbor: write down the pair (x, y), in that order.
3. Delete vertex x and edge xy from T to get an $(n-1)$-vertex tree $T - x$. To write the rest of the sequence, go back to step 1, but with tree $T - x$ in place of T.

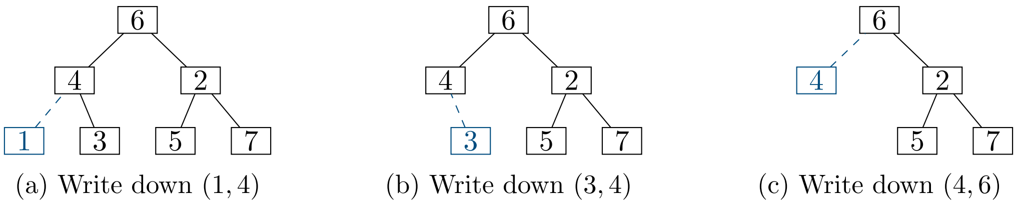
We will call the result of this algorithm a **deletion sequence** for T. (This is not a universally recognized term, but simply the term I will use in this chapter to explain our counting strategy.) As an illustration of the technique, Figure 11.2 shows how, starting with the tree in Figure 11.2a, we determine that its deletion sequence is

$$(1,4),\ (3,4),\ (4,6),\ (5,2),\ (6,2),\ (2,7).$$

Question: Is the deletion sequence of a tree a unique encoding of that tree?

Answer: Yes, it is. We can recover the tree from its deletion sequence, because the deletion sequence is after all a list of edges. Moreover, each tree only has one deletion sequence, because the deletion sequence is computed by an algorithm with no freedom at any step.

There is a great deal of redundancy in the deletion sequence of a tree. Before proving a general result about it, let's explore a few examples. In all of these, I will erase some numbers from the deletion sequence we just constructed and ask how they can be filled back in to get a valid deletion sequence. Of course, one way to fill in the number is to put back the number we erased, getting back the deletion sequence we started with. However, we want to know if there are any other deletion sequences that have a different value in that blank!



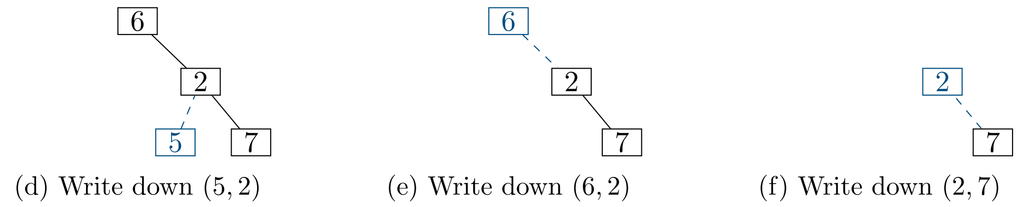


Figure 11.2: Finding the deletion sequence of a tree

Question: In the incomplete deletion sequence

$$(1,4),\ (3,4),\ (4,6),\ (5,2),\ (6,2),\ (2,\_),$$

how many ways are there to fill in the blank?

Answer: The number in the blank can only be 7.

The reason is that vertex 7 will never be deleted as the smallest leaf of the tree: there are always at least two leaves, one of which is smaller than 7. Therefore it is the last vertex remaining, and will always occupy the last position in the deletion sequence.

Question: In the incomplete deletion sequence

$$(1,4),\ (3,4),\ (\_,6),\ (5,2),\ (6,2),\ (2,7),$$

how many ways are there to fill in the blank?

Answer: The number in the blank can only be 4.

The reason is that vertices $1, 2, 3, 4, 5, 6$ must all be eventually deleted. The five complete ordered pairs tell us when we deleted vertices 1, 2, 3, 5, and 6, so the incomplete ordered pair must tell us when we deleted vertex 4.

Question: In the incomplete deletion sequence

$$(1,4),\ (3,4),\ (\_,6),\ (\_,2),\ (6,2),\ (2,7),$$

how many ways are there to fill in the two blanks?

Answer: Once again, the answer is uniquely determined: the blanks must contain 4 and 5, in that order.

By the same reasoning as above, we know that 4 and 5 must go in those two blanks in some order. Since neither number appears later in the deletion sequence, we know that none of their neighbors are deleted later: after the first two steps, both vertices 4 and 5 are leaves. Since 4 is a smaller number, it will be the leaf deleted first.

But all of these questions are thinking too small: we can take the incomplete sequence

$$(\_,4),\ (\_,4),\ (\_,6),\ (\_,2),\ (\_,2),\ (\_,\_)$$

and fill in all 7 blanks in a unique way!

First of all, we know that the first six blanks are the numbers 1 through 6, while the last blank is 7. The order of the numbers 1 through 6 is not known yet, but even without knowing the order, we know how many times each number appears in the deletion sequence, in total: the number of times we see it in the incomplete sequence, plus 1.

This tells us the degree of every vertex:

x	1	2	3	4	5	6	7
$\deg(x)$	1	3	1	3	1	2	1

Why is this helpful? Because it tells us that at the beginning of the algorithm to generate the deletion sequence, the leaves were 1, 3, 5, and 7. Of these, 1 is the smallest, so it must be the first leaf deleted: the first pair is $(1,4)$.

From there, we can deduce that after vertex 1 is deleted, the degrees of the vertices were as follows:

x	2	3	4	5	6	7
$\deg(x)$	3	1	2	1	2	1

The leaves were 3, 5, and 7, of which 3 is the smallest. Therefore the second pair must be $(3,4)$.

If we keep going in this manner, we can reconstruct the entire deletion sequence, because we can determine the degree of each remaining vertex at each step of the algorithm. Just 5 of the 12 numbers in the deletion sequence were necessary!

11.3 Prüfer codes

Our strategy in the preceding section generalizes fully. We can summarize the properties we use in the following two properties of a sequence $(a_1,b_1),(a_2,b_2),\dots,(a_{n-1},b_{n-1})$:

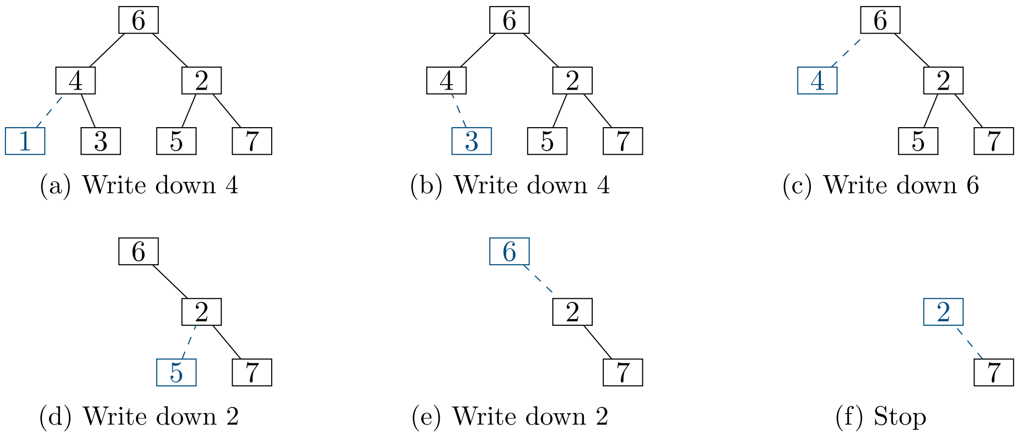


Figure 11.3: Finding the Prüfer code of a tree

For every k from 1 to $n-1$, the number a_k is the smallest positive number not contained in the set $\{a_1, a_2, \ldots, a_{k-1}\} \cup \{b_k, b_{k+1}, \ldots, b_{n-2}\}$. (11.1)

$$b_{n-1} = n. \tag{11.2}$$

Lemma 11.1. *If a sequence* $(a_1, b_1), (a_2, b_2), \ldots, (a_{n-1}, b_{n-1})$ *is the deletion sequence of a tree with vertices* $1, 2, \ldots, n$*, then it satisfies properties* (11.1) *and* (11.2).

Proof. The tree at every stage of the algorithm that generates the deletion sequence has at least two leaves, so the smallest leaf will never be vertex n. Therefore vertex n will never be the deleted leaf, so it will be the last vertex remaining: $b_{n-1} = n$. This proves (11.2).

Meanwhile, $a_1, \ldots, a_{n-1}$ are a permutation of $1, 2, \ldots, n-1$, representing the order in which the other vertices are deleted. After the first $k-1$ stages of the algoritm that generates the deletion sequence, the tree that remains has edges (a_k, b_k) through (a_{n-1}, b_{n-1}). Vertices in the set $\{a_1, a_2, \ldots, a_{k-1}\}$ are not present in this tree; they have already been deleted.

The remaining vertices of the tree appear once in the set $\{a_k, a_{k+1}, \ldots, a_{n-1}, b_{n-1}\}$. They have degree 1 if this is their only appearance: if they do not appear in the set $\{b_k, b_{k+1}, \ldots, b_{n-2}\}$. Vertex a_k is the smallest leaf remaining at this stage, so it is the smallest positive number not contained in the set $\{a_1, a_2, \ldots, a_{k-1}\} \cup \{b_k, b_{k+1}, \ldots, b_{n-2}\}$. This proves (11.1). □

Since the numbers determined by Lemma 11.1 can be deduced from the others, they are not necessary to recover the tree. Therefore, instead of recording the deletion sequence of a tree, it is enough to record the sequence $(b_1, b_2, \ldots, b_{n-2})$. This sequence is called the **Prüfer code** of the tree, named after Heinz Prüfer, who proposed Prüfer codes as a method of counting labeled trees in 1918 [7].

It is worth mentioning that the Prüfer code of a tree can be directly computed using an abbreviated version of the algorithm that computed the deletion sequence. The only two differences are that instead of writing down the pair (x, y), we only write down the vertex y, and that we stop when there are 2 vertices and 1 edge left. Figure 11.3 illustrates this on an example.

Question: Does the Prüfer code of a tree uniquely encode that tree?

Answer: Yes: from the Prüfer code, we can recover the deletion sequence using properties (11.1) and (11.2), and the deletion sequence simply tells us the edges of the tree.

Question: Is this enough to count the labeled trees on n vertices?

Answer: No: we need to know whether all possible sequences $(b_1, b_2, \dots, b_{n-2})$ are valid Prüfer codes, or whether there are some constraints on these numbers.

In fact, there are no further constraints: every sequence $(b_1, b_2, \dots, b_{n-2})$, where each term is an element of $\{1, 2, \dots, n\}$, is the Prüfer code of a tree with vertex set $\{1, 2, \dots, n\}$. To prove this, the most important claim we have not yet shown is that (11.1) and (11.2) are not just properties every deletion sequence has: they are properties only a deletion sequence can have.

Lemma 11.2. *If a sequence $(a_1, b_1), (a_2, b_2), \dots, (a_{n-1}, b_{n-1})$ in which every term (a_i, b_i) is a pair of numbers from 1 to n satisfies properties* (11.1) *and* (11.2)*, then it is the deletion sequence of a tree with vertex set $\{1, 2, \dots, n\}$.*

Proof. Let's make some initial observations. First, since $a_k \notin \{a_1, a_2, \dots, a_{k-1}\}$, the numbers $a_1, a_2, \dots, a_{n-1}$ are distinct. They are all smaller than n, since each is the smallest positive number not contained among at most $n-2$ options; therefore, they are a permutation of $\{1, 2, \dots, n-1\}$.

Second, consider b_k for $1 \le k \le n-2$. Since none of $a_1, a_2, \dots, a_k$ can equal b_k, but b_k is an integer from 1 to n, we know that it must be an element of $\{a_{k+1}, \dots, a_{n-1}, n\}$. From this observation, it follows that we can define a graph T_k with $V(T_k) = \{a_k, a_{k+1}, \dots, a_{n-1}, n\}$ and $E(T_k) = \{a_k b_k, a_{k+1} b_{k+1}, \dots, a_{n-1} b_{n-1}\}$: each edge in $E(T_k)$ really does have both endpoints in $V(T_k)$.

We are now ready to proceed with the proof. The graph T_k is not just any graph: it is a tree in which the leaf with the smallest number is a_k. We will prove this by an induction that starts with T_{n-1} and ends with T_1.

In the base case, T_{n-1} is the graph with vertices $\{a_{n-1}, n\}$ and edge $a_{n-1}b_{n-1}$; since $b_{n-1} = n$, this is an edge between T_{n-1}'s two vertices, so T_{n-1} is a tree. Both a_{n-1} and b_{n-1} are leaves, but since $b_{n-1} = n$ and $a_{n-1} \ne b_{n-1}$, a_{n-1} must be the smaller leaf.

Next, for some positive $k < n-1$, suppose T_{k+1} is a tree. The only difference between T_k and T_{k+1} is that we add vertex a_k and edge $a_k b_k$. By our first observation, $a_k \notin V(T_{k+1})$, so a_k is a leaf of T_k. Since T_{k+1} is a tree, it is acyclic, so T_k cannot contain any cycles not using vertex a_k; it cannot have any cycles using a_k, either, because a_k has degree 1. Therefore T_k is still acyclic; we know it has $n-k+1$ vertices and $n-k$ edges, so it is a tree by condition 5 of Theorem 10.2.

Each a_i for $i < k$ is not yet a vertex of T_k, by our first observation. Each b_i for $i \ge k$ appears a second time as a_j for $j > i$, by our second observation; so it is the endpoint of at least two edges of T_k, and is not a leaf. Among the remaining positive integers, a_k is the smallest; therefore in particular it is the smallest leaf of T_k. This completes the induction.

From this claim, it follows that the deletion sequence algorithm, when encountering tree T_k, will write down the pair (a_k, b_k) and delete a_k, then either go on to tree T_{k+1} or (if $k = n-1$) stop. In particular, if we start with tree T_1, the deletion sequence algorithm will proceed through the trees $T_2, T_3, \dots, T_{n-1}$ and write down the sequence

$$(a_1, b_1), (a_2, b_2), \dots, (a_{n-1}, b_{n-1}),$$

which is exactly what we wanted to show. □

With this setup complete, we are ready to complete the count of labeled trees. This theorem is known as Cayley's formula after Arthur Cayley, whom we already know as the mathematician that came up with the term "tree". (So, you see, Prüfer's argument was not the first to be found. However, as it sometimes happens, Cayley was not the first, either; the formula was first proven by Carl Wilhelm Borchardt in 1860 [2].)

Theorem 11.3 (Cayley's formula)**.** *There are n^{n-2} trees with vertex set $\{1, 2, \dots, n\}$.*

Proof. Each tree with vertex set $\{1, 2, \dots, n\}$ has a Prüfer code $(b_1, b_2, \dots, b_{n-2})$ whose elements are integers between 1 and n, uniquely defined by the deletion sequence algorithm. Therefore the number of trees with vertex set $\{1, 2, \dots, n\}$ is exactly the number of possible Prüfer codes.

Moreover, for every such sequence $(b_1, b_2, \dots, b_{n-2})$, we can apply (11.1) to determine a_1, then a_2, and so on through a_{n-1}, as long as we go in that order: each a_k will be the smallest positive integer not contained in a set we've already entirely determined. We can also set $b_{n-1} = n$. Now, the sequence

$$(a_1, b_1), (a_2, b_2), \dots, (a_{n-1}, b_{n-1})$$

is a deletion sequence of a tree with vertex set $\{1, 2, \dots, n\}$, by Lemma 11.2, and therefore $(b_1, b_2, \dots, b_{n-2})$ is the Prüfer code of that tree. This shows that every sequence $n-2$ integers from 1 to n is a valid Prüfer code. There are exactly n^{n-2} such sequencess, since there are n options for each of the numbers b_1 through b_{n-2}, so the number of trees with vertex set $\{1, 2, \dots, n\}$ is also n^{n-2}. □

11.4 Working with Prüfer codes

There are a few more difficult questions to ask about Prüfer codes, but let's first pause to make sure that we can return from the abstract results to concrete claims. Take a Prüfer code like $(2, 1, 2, 5, 4)$. How do we turn it back into a tree?

Let's write down what we know about the deletion sequence of that tree, even if there's still some blanks to be filled in:

$$(\_, 2), (\_, 1), (\_, 2), (\_, 5), (\_, 4), (\_, \_).$$

A Prüfer code is always a sequence of $n-2$ numbers from 1 to n; since we started with 5 numbers, their values will range from 1 to 7. We fill in the blanks from left to right.

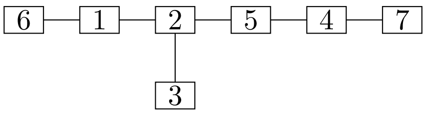


Figure 11.4: The tree with Prüfer code $(2, 1, 2, 5, 4)$.

For the first blank, which is a_1, we use (11.1). There are no a-terms before a_1; however, a_1 needs to be distinct from $\{b_1, b_2, b_3, b_4, b_5\} = \{1, 2, 4, 5\}$. The smallest integer not on this list is 3, so we set $a_1 = 3$:

$$(3, 2), (\_, 1), (\_, 2), (\_, 5), (\_, 4), (\_, \_).$$

We continue to use (11.1) for the next blank, which is a_2. Here, the excluded values are a_1, b_2, b_3, b_4, b_5 or $3, 1, 2, 5, 4$. We fill in the blank with the first integer not on this list, which is 6:

$$(3, 2), (6, 1), (\_, 2), (\_, 5), (\_, 4), (\_, \_).$$

We go on in this way. Some people prefer to arrange the entries in a $2 \times (n - 1)$ table, where each a_i entry (in the first, initially empty row) must be distinct from everything to its left, as well as everything below it and to the right:

$$\begin{array}{c|c|c|c|c|c} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 \\ \hline b_1 & b_2 & b_3 & b_4 & b_5 & b_6 \end{array} \quad = \quad \begin{array}{c|c|c|c|c|c} 3 & 6 & ? & & & \\ \hline 2 & 1 & 2 & 5 & 4 & \end{array}$$

Regardless, we fill in $a_3 = 1$ (the smallest value not among $3, 6, 2, 5, 4$), then $a_4 = 2$ (the smallest value not among $3, 6, 1, 5, 4$), then $a_5 = 5$ (the smallest value not among $3, 6, 1, 2, 4$), then $a_6 = 4$ (the smallest value not among $3, 6, 1, 2, 5$). All this is done using (11.1); then, the last blank is $b_6 = 7$ by (11.2). The completed deletion sequence is

$$(3, 2), (6, 1), (1, 2), (2, 5), (5, 4), (4, 7).$$

This sequence tells us the edges of the tree, and if we like, we can draw a diagram such as the one in Figure 11.4.

That being said, it is not too often that we find ourselves needing to actually convert a Prüfer code back into a tree: it only matters for the proof of Theorem 11.3 that we can, in principle, do it.

This does mean that Prüfer codes have no use beyond the proof of Theorem 11.3. The Prüfer code of a tree actually contains a bit of information about the tree, which we can use to solve more complicated counting problems. For example:

Proposition 11.4. *In the Prüfer code of a tree where* $\deg(x) = k$, *the number* x *appears* $k - 1$ *times.*

Proof. When we fill in the blanks in the sequence

$$(\_\_, b_1), (\_\_, b_2), \dots, (\_\_, b_{n-2}), (\_\_, \_\_)$$

we use each of the numbers $1, 2, \dots, n$ once. The number n is going to fill in the second blank of the last pair, and the numbers in the first blanks are a permutation of $1, 2, \dots, n - 1$.

Therefore if a number x appears $k-1$ times in the Prüfer code, it appears k times in the deletion sequence

$$(a_1, b_1), (a_2, b_2), \dots, (a_{n-1}, b_{n-1}).$$

But the deletion sequence is just a particular way to write down the edges of the tree, so if x appears in the deletion sequence k times, then it is the endpoint of k edges, which is just another way of saying that $\deg(x) = k$. □

For example, even before we turned the Prüfer code $(2, 1, 2, 5, 4)$ back into the tree shown in Figure 11.4, we could have known the rough structure of the tree:

- Vertices 3, 6, and 7 are leaves, because they do not appear in the Prüfer code at all.
- Vertices $1, 4, 5$ appear once, and therefore have degree 2.
- Vertex 2 appears twice, and therefore has degree 3.

Question:	What can we say about the shape of such a tree, knowing only the degrees of the vertices?
Answer:	Such a tree must consist of three paths starting at vertices 3, 6, 7 and converging at vertex 2.

Here is a quick example of using Proposition 11.4 to solve a counting problem:

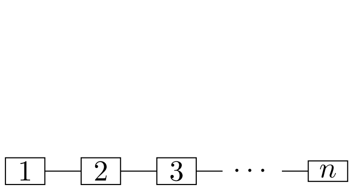
Corollary 11.5. *There are $(n-1)^{n-2}$ trees with vertex set $\{1, 2, \dots, n\}$ in which vertex 1 is a leaf.*

Proof. By Proposition 11.4, vertex 1 is a leaf (has degree 1) if and only if the number 1 never appears in the Prüfer code. There are $(n-1)^{n-2}$ such codes: the code is a sequence with $n-2$ terms, each of which is now restricted to one of the $n-1$ values in the set $\{2, 3, \dots, n\}$. Therefore there are $(n-1)^{n-2}$ such trees. □

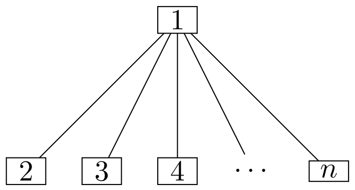
11.5 Counting unlabeled trees

Now that we've counted labeled trees on n vertices, we can try to say something about the unlabeled trees on n vertices as well, but it will be difficult.

The unlabeled trees can be thought of as equivalence classes of the n^{n-2} labeled trees. Each equivalence class is a set of trees that are isomorphic, but differently labeled. Sometimes such problems can be handled by a simple division, but only if we're very lucky and the equivalence classes all have the same size. Here, that is far from true.



(a) An n-vertex path



(b) An n-vertex-star

Figure 11.5: Two labeled trees with very different structures

Question: Figure 11.5a shows one labeled tree on n vertices: an n-vertex path. How many labeled trees on n vertices are isomorphic to this path?

Answer: There are $n!$ ways to put the vertices in order from left to right along the path. However, the graph does not know a "left" and a "right": if you reverse the sequence, you get the same graph, drawn in reverse. Therefore there are $\frac{1}{2}n!$ paths with vertex set $\{1, 2, \dots, n\}$.

Question: Figure 11.5b shows another labeled tree on n vertices: an n-vertex star. How many labeled trees on n vertices are isomorphic to this star?

Answer: As soon as we choose which vertex in the set $\{1, 2, \dots, n\}$ is the vertex of degree $n-1$ at the center of the star, the graph is completely determined, so there are n labeled n-vertex stars.

If all n-vertex trees were like the tree in Figure 11.5a, then every equivalence class would have $\frac{1}{2}n!$ elements, and the number of unlabeled trees would be the quotient $n^{n-2}/(\frac{1}{2}n!)$. If instead all n-vertex trees were like the tree in Figure 11.5b, then every equivalence class would have n elements, and the number of unlabeled trees would be $\frac{n^{n-2}}{n}$ or n^{n-3}.

In reality, neither of these extremes is the case. The truth is somewhere in the middle; but it is more like the first answer than the second. Why? Well, the reason that there are very few different trees isomorphic to the star in Figure 11.5b is that the star graph has a lot of symmetry. Most large trees are not nearly as symmetric, so most equivalence classe are pretty large.

A result known as Stirling's formula says that, very approximately, $n!$ grows like $(\frac{n}{e})^n$, where $e \approx 2.718$ is Euler's number.[1] If we plug this into the quotient $n^{n-2}/(\frac{1}{2}n!)$, we get an estimate of $2e^n/n^2$ for the number of unlabeled trees. This is not the true growth rate, but it is the right type of growth: the number of unlabeled trees really does grow exponentially. That growth rate was first precisely analyzed in 1937 by Hungarian-American mathematician George Pólya, who described it as an exponential c^n with $c \approx 2.9557$, divided by a polynomial factor.

No exact formula is known. In the Online Encyclopedia of Integer Sequences, the number of n-vertex unlabeled trees can be found in one of the very first sequences: sequence A000055 [8]. The first few terms are

$$1, 1, 1, 1, 2, 3, 6, 11, 23, 47, 106, 235, \dots$$

[1]See the second problem at the end of this chapter if you want to know more digits of this constant, but prepare to do some work, first.

The initial 1's in this sequence correspond to $n = 0$[2] through $n = 3$, where only one n-vertex tree is possible. (For $n = 4$, we have two options: the 4-vertex path, and the 4-vertex star.)

11.6 Practice problems

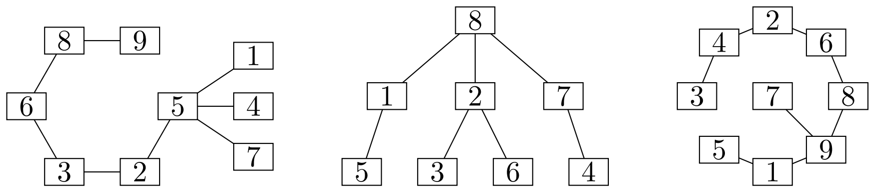
1. Find the trees with the following Prüfer codes:

   a) $(1, 1, 1, 1, 1)$.

   b) $(1, 2, 3, 4, 5, 6)$.

   c) $(3, 1, 4, 1, 5, 9, 2)$.

2. Find the Prüfer codes of the following trees:

   (One of these Prüfer codes gives you my birth date. Another is the first few digits of Euler's number e. Another tells you the phone number to call to reach Ghostbusters.)

3. Find all 16 trees with vertices $\{1, 2, 3, 4\}$. (Prüfer codes are not very useful here.)

4. What is the Prüfer code of the tree in Figure 11.5a, and what is the general form of a Prüfer code for a tree isomorphic to this one?

5. What is the Prüfer code of the tree in Figure 11.5b, and what is the general form of a Prüfer code for a tree isomorphic to this one?

6. Use Prüfer codes and Proposition 11.4 to count:

   a) The number of trees with vertex set $\{1, 2, \ldots, n\}$ in which vertex 1 has degree 3.

   b) The number of trees with vertex set $\{1, 2, \ldots, n\}$ in which vertex 1 has degree $n - 2$.

   c) The number of trees with vertex set $\{1, 2, \ldots, n\}$ in which all vertices except vertex 1 and 2 have degree 1.

   For parts (b) and (c), think about how you would count them without using Prüfer codes.

7. Use Corollary 11.5 to find the average number of leaves in an n-vertex labeled tree.

8. Prove that if a tree has maximum degree d, then it has at least d leaves:

   a) Using Prüfer codes and Proposition 11.4.

   b) Using ideas from the previous chapter.

[2] I actually disagree with the OEIS on the initial value; I do not believe there is a 0-vertex tree. Even if we allow 0-vertex graphs to be exist, they should surely have 0 edges, but an n-vertex tree should have $n - 1$ edges. My opinion, which does not really affect anything but definitions and initial terms, is that the 0-vertex graph exists but is not connected.

9. Each possible edge e of the complete graph with vertex set $\{1, 2, \ldots, n\}$ is contained in $f(n)$ of the n^{n-2} trees with this set of vertices. By symmetry, $f(n)$ is the same for any edge.

   a) Determine and prove a formula for $f(n)$.

   b) Let K_n^- be the complete graph with a single edge deleted. (Up to isomorphism, it doesn't matter which edge.) Find the number of spanning trees that K_n^- has, in terms of n.

Bibliography

[1] Berkeley Math Circle. *7th Bay Area Mathematical Olympiad.* 2005. URL: https://www.bamo.org/archives/examfiles/bamo2005examsol.pdf (visited on 09/23/2025).

[2] Carl Wilhelm Borchardt. “Über eine Interpolationsformel für eine Art symmetrischer Functionen und über deren Anwendung”. In: *Mathematische Abhandlungen der Königlichen Akademie der Wissenschaften zu Berlin* (1860), pp. 1–20. URL: https://archive.org/details/abhandlungenderk1860deut/page/n245.

[3] Arthur Cayley. “On the theory of the analytical forms called trees”. In: *Philosophical Magazine* 13 (1857), pp. 172–176. URL: https://rcin.org.pl/Content/173708/PDF/WA35_185808_12807-3_art-45.pdf.

[4] Joseph B. Kruskal. “On factors with all degrees odd”. In: *Graphs and Combinatorics* 1 (1985), pp. 111–114. DOI: 10.1007/BF02582935.

[5] Joseph B. Kruskal. “On the shortest spanning subtree of a graph and the traveling salesman problem”. In: *Proceedings of the American Mathematical Society* 7.1 (1956), pp. 48–50. DOI: 10.2307/2033241.

[6] P.C. Li and Michel Toulouse. “Maximum leaf spanning tree problem for grid graphs”. In: *Journal of Combinatorial Mathematics and Combinatorial Computing* 73 (2010), p. 181. URL: https://combinatorialpress.com/jcmcc-articles/volume-073/maximum-leaf-spanning-tree-problem-for-grid-graphs/.

[7] Heinz Prüfer. “Neuer Beweis eines Satzes über Permutationen”. In: *Arch. Math. Phys* 27 (1918), pp. 742–744.

[8] N.J.A. Sloane. *Sequence A001292 in the On-Line Encyclopedia of Integer Sequences.* URL: https://oeis.org/A000055 (visited on 09/23/2025).