Math 3322: Graph Theory[1] Mikhail Lavrov

Lecture 12: Counting trees

September 19, 2024 *Kennesaw State University*

1 Counting problems about trees

1.1 The labeled tree problem

How many n-vertex trees are there?

There are several ways we could ask that question. For example, we could ask: how many trees on n vertices are there, up to isomorphism? This might be one of the most natural versions of the question, but we're not going to ask it, because it's too hard.

Instead, we will ask the following question. Suppose we fix a set $V = \{v_1, v_2, \ldots, v_n\}$. How many trees are there with vertex set V? Equivalently: how many spanning trees does K_n have? Here, we count trees separately if they have a different set of edges, even if they are isomorphic. We call this the problem of counting **labeled** trees on n vertices, because the "labels" $v_1, v_2, \ldots, v_n$ matter.

1.2 Counting encodings

A more fundamental question is this: in general, how do you prove the answer to a counting problem like this one?

We can think about ways to concisely write down all the data telling us which tree we have. If we have an encoding from which we can unambiguously recover the tree we started with, then counting trees is just as easy as counting encodings.

Here is one way we could encode a tree with vertex set $\{v_1, v_2, \ldots, v_n\}$. We could list all the possible edges $v_1v_2, v_1v_3, v_1v_4, \ldots, v_{n-1}v_n$ in some order we've settled on in advance. Then, for each edge, write 1 if the edge is present in the tree, and 0 if it's absent. For example, suppose that when $n = 4$, we settle on the order $v_1v_2, v_1v_3, v_1v_4, v_2v_3, v_2v_4, v_3v_4$. Then the path P_4 (a tree with edges $\{v_1v_2, v_2v_3, v_3v_4\}$ will be encoded by 100101.

There are $\binom{n}{2}$ binary digits in this encoding; each has 2 options, so there are $2^{\binom{n}{2}}$ possible strings. Unfortunately, this doesn't immediately tell us that there are $2^{\binom{n}{2}}$ trees, because not every string gives us a tree. For example, the string 111111 would give us the complete graph K_4 instead. Actually, this just tells us that there are $2^{\binom{n}{2}}$ subgraphs of K_n.

Here is another possible encoding. Go through the edges of the tree in some order; for every edge v_iv_j, write down the numbers i and j. This gives us a sequence of numbers; for example, P_4 would be written down as $1, 2, 2, 3, 3, 4$. (We could group the edges together, writing down $(1, 2), (2, 3), (3, 4)$, but this shouldn't matter for counting.)

[1] This document comes from an archive of the Math 3322 course webpage: http://misha.fish/archive/3322-fall-2024

Each number is between 1 and n, and there are $2n-2$ numbers: 2 numbers for each of the $n-1$ edges. As a result, there are n^{2n-2} possible encodings. Unfortunately, this does not mean that there are n^{2n-2} possible trees! There are two problems:

- Each edge can be written down in 2 possible orders, and the edges themselves can be reordered in $(n-1)!$ ways. So there are $(n-1)!\,2^{n-1}$ different encodings of every tree; we should divide by this number.
- But even that doesn't help. Unfortunately, some encodings are not valid. For example, another way to choose 6 numbers between 1 and 4 is to choose $(1,2),(2,3),(1,3)$. The graph this represents has vertices $\{v_1,v_2,v_3,v_4\}$ and edges $\{v_1v_2,v_2v_3,v_1v_3\}$; it is not a tree.

So n^{2n-2} is also an overestimate of the number of trees, though it is closer to the truth.

2 Prüfer codes

Our last approach did not work, but it is promising. We are going to try to solve the problems with it by eliminating the redundancy in the encoding. That is, we will:

1. Establish conventions about which order we write things down in, so that each tree gets only one possible encoding.
2. Get rid of information that we could deduce from other things we've already written down.

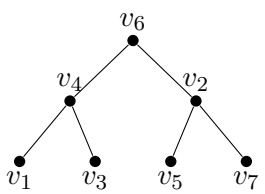
In the end, if it's also true that every encoding corresponds to a tree, then we'll know that the number of encodings is equal to the number of trees.

2.1 Writing down all the edges

For step 1, we will use a fact we established in the previous lecture: every tree with $n \geq 2$ vertices has at least one leaf, and if we delete that leaf, we get an $(n-1)$-vertex tree. So here is the method we will use to write down all the edges in a different order:

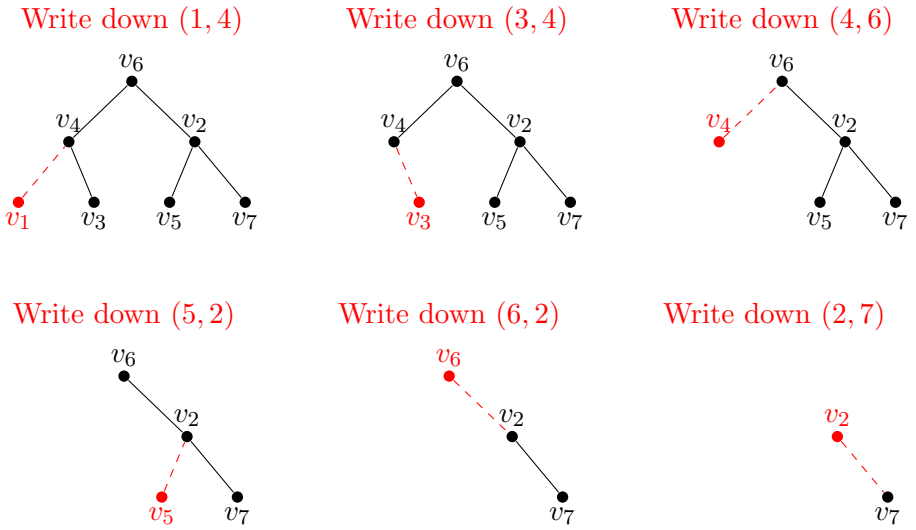
1. Let v_i be a leaf of the tree T. Actually, there will always be at least two leaves, so to avoid making arbitrary choices, let v_i be the leaf with the smallest value of i.
2. Let v_j be the only neighbor of v_i in T. Write down the pair (i,j) to represent the edge v_iv_j.
3. Delete vertex v_i and edge v_iv_j form T to get an $(n-1)$-vertex tree T'. If $n-1>1$, then go back to step 1 with T' in place of T. (If $n-1=1$, stop; we've written down all the edges.)

Here is an example. Consider the following tree:



We will delete the vertices $v_1, v_3, v_4, v_5, v_6, v_2$ in that order, and we will write down the encoding $(1,4)$, $(3,4)$, $(4,6)$, $(5,2)$, $(6,2)$, $(2,7)$.

Here's the step-by-step breakdown of how we get the encoding:



Let's make up a name for an encoding like $(1,4)$, $(3,4)$, $(4,6)$, $(5,2)$, $(6,2)$, $(2,7)$: call it a "deletion sequence", because it is a sequence of edges we delete from the tree. (This is not an official term.)

2.2 Eliminating redundancy

First, here is an example of the redundancy. Suppose I erase one number in the deletion sequence: I leave you with

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

Can we recover the number that was there? Yes! As we delete vertices from this tree, each vertex will eventually be a leaf, and each vertex except for v_7 will be deleted. So the first numbers in the pairs should be $1,2,3,4,5,6$ in some order. We are missing 4, so 4 must go in that blank.

Here is a more complicated example. Suppose I erase two numbers, and only give you the following information:

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

By our earlier reasoning, we know that the two blanks must contain 4 and 5 in *some* order. You might think that either order could work.

However, looking ahead, we see that there are no later edges that contain 4 or 5. So after the edges v_1v_4 and v_3v_4 are deleted, both v_4 and v_5 must be leaves. Our rule is to pick the leaf v_i with the *smallest* i at every step. So we must have picked v_4 before v_5, and therefore the labels are 4 and 5 in that order.

In fact, even if I erase the first number in *every* pair, we can fill in the blanks in

$$(\_,4),(\_,4),(\_,6),(\_,2),(\_,2),(\_,7).$$

We will give a systematic rule for this later, but here's the idea. Since the six blanks must be $1,2,3,4,5,6$ in some order, we know how many times each number shows up, so we can deduce the

degree sequence

$$\deg(v_1) = 1,\ \deg(v_2) = 3,\ \deg(v_3) = 1,\ \deg(v_4) = 3,\ \deg(v_5) = 1,\ \deg(v_6) = 2,\ \deg(v_7) = 1.$$

This means that from the start, v_1, v_3, v_5, v_7 are leaves. We always delete the smallest leaf, so the first blank is 1, and the second blank is 3. At that point, v_4 has lost two neighbors, so it is also a leaf, and 4 is smaller than 5 or 7, so it goes in the third blank. If we keep going like this, we'll end up writing 5, then 6, then 2.

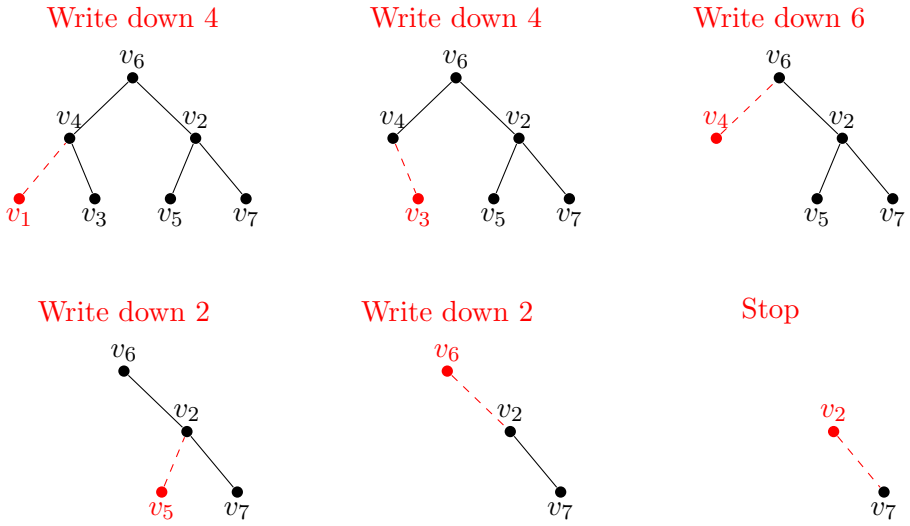
There is one more element of redundancy. The second coordinate of the *last* pair is always 7 (in general, n), because v_n is always the last vertex left. So we don't need to write it down, either.

The remaining numbers (44622 in this example) are the **Prüfer code** of the tree. Given the Prüfer code 44622, we can fill in the blanks in

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

and recover the edges of the tree.

If we wanted to directly write down the Prüfer code 44622, we'd follow the same edge-deletion algorithm that we did previously, but we'd write down less information. Here's how it would go:



To describe how we recover the deletion sequence from the Prüfer code in general, we need to prove two lemmas:

Lemma 2.1. *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 $v_1, v_2, \ldots, v_n$, then it has the following properties:*

1. *For every k from 1 to $n-1$, the number a_k is the smallest positive number not contained in the set $\{a_1, \ldots, a_{k-1}\} \cup \{b_k, \ldots, b_{n-1}\}$.*
2. *The last number b_{n-1} is n.*

Proof. After the first $k-1$ steps of the algorithm, which of the vertices among $\{v_1, v_2, \ldots, v_{n-1}\}$ are leaves? Vertex v_i is a leaf of that tree exactly when (1) i does not appear in $\{a_1, \ldots, a_{k-1}\}$, or else v_i would have already been deleted, and (2) i does not appear in $\{b_k, \ldots, b_{n-1}\}$, or else v_i has other vertices "hanging off of it".

We delete the leaf with the smallest index, so a_k is the smallest number that satisfies both properties.

The second property is easy: the operation of deleting the smallest leaf never deletes vertex v_n, so v_n must be the last vertex left in the tree, and therefore we write down n as b_{n-1}. □

The reason to prove these two properties in particular is the following lemma:

Lemma 2.2. *Suppose that a sequence $(a_1, b_1), (a_2, b_2), \ldots, (a_{n-1}, b_{n-1})$, where every element is a pair of numbers from 1 to n, has the two properties in Lemma 2.1. Then it is the deletion sequence of a tree with vertices $v_1, v_2, \ldots, v_n$.*

Proof. Some observations, first. The first half of property 1, which says that $a_k \notin \{a_1, a_2, \ldots, a_{k-1}\}$, implies that the $n-1$ numbers $a_1, a_2, \ldots, a_{n-1}$ are all distinct; also, none of them are equal to $b_{n-1} = n$, so they are a permutation of $\{1, 2, \ldots, n-1\}$. Also, since none of $a_1, a_2, \ldots, a_k$ are allowed to equal b_k, but b_k is an integer from 1 to n, we must have $b_k \in \{a_{k+1}, \ldots, a_{n-1}, n\}$.

Now, let's work backwards. Start from a tree with one vertex, v_n, and no edges. Then on step k, where k counts *backwards* from $n-1$ to 1, we add an edge $v_i v_j$, where $(i, j) = (a_k, b_k)$.

Because $j = b_k \in \{a_{k+1}, \ldots, a_{n-1}, n\}$, vertex v_j is already in the graph we've built. Because $i = a_k \notin \{b_k, \ldots, b_{n-1}\}$ (by property 1) and $i = a_k \notin \{a_{k+1}, \ldots, a_{n-1}\}$ (by our earlier observations), vertex v_i is new to the graph. So we're adding a new leaf vertex to our graph, which means we will continue to have a tree at each step.

A vertex v_i, with $1 \le i \le n-1$, appears in the tree after step k if $i \in \{a_k, \ldots, a_{n-1}\}$ (equivalently, if $i \notin \{a_1, \ldots, a_{k-1}\}$. Moreover, v_i is a leaf exactly when $i \notin \{b_k, \ldots, b_{n-1}\}$: when we didn't add new vertices adjacent to $v-$i after adding v_i. These are exactly the conditions in the lemma, and since a_k is the smallest number satisfying them, it is the smallest number of any leaf. Therefore the edge $v_i v_j$ with $(i, j) = (a_k, b_k)$ is exactly the edge we delete to get the deletion sequence, proving the lemma. □

2.3 Reconstructing trees from Prüfer codes

We know how to take a tree and write down a Prüfer code like "44622" from it. Now let's figure out how to work backwards. Suppose that we are given an arbitrary code $b_1 b_2 \ldots b_{n-2}$. How can we find the original 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:

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

We immediately fill in the last blank, b_{n-1}, with the value n: that's always the second number in the last pair. Now we've got

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

Next, we fill in the other blanks with some values $a_1, a_2, \ldots, a_{n-1}$. We go from left to right, and the rule that we follow is exactly the rule in Lemma 2.1 and Lemma 2.2: we pick a_k to be the smallest positive integer not contained in the set $\{a_1, \ldots, a_{k-1}\} \cup \{b_k, \ldots, b_{n-1}\}$.

How do we know this always gives us a number from 1 to n? Well, there's n possibilities in that range. At most $k-1$ of the are eliminated by ruling out $a_1, \ldots, a_{k-1}$, and at most $n-k$ are eliminated by ruling out $b_k, \ldots, b_{n-1}$. This still leaves at least one possibility, because $n - (k-1) - (n-k) = 1$.

So we can always follow this procedure. What's more, once we're done, the resulting sequence

$$(a_1, b_1), (a_2, b_2), \ldots, (a_{n-2}, b_{n-2}), (a_{n-1}, b_{n-1})$$

really is the deletion sequence of some tree: that's what Lemma 2.2 tells us! Also, the only rules we used to fill in the blanks are the rules that came from Lemma 2.1, which *must* hold for any deletion sequence. So we know that we've reconstructed the *unique* deletion sequence that corresponds to the Prüfer code $b_1 b_2 \ldots b_{n-2}$.

3 Applications of Prüfer codes

3.1 Counting trees with Prüfer codes

Let's wrap up the story of Prüfer codes by showing how finding a way to encode trees lets us count trees.

Theorem 3.1 (Cayley)**.** *There are exactly n^{n-2} labeled n-vertex trees.*

Proof. A Prüfer code of an n-vertex tree is a sequence $b_1 b_2 \ldots b_{n-2}$, where each element $b_1, b_2, \ldots, b_{n-2}$ is an integer from 1 to n. This means that there are exactly n^{n-2} possible Prüfer codes.

Given a Prüfer code, not only can we always "fill in the blanks" to recover a deletion sequence, but the only tools we need to do so are the two properties in Lemma 2.1, which *must* hold for every deletion sequence. This means that there is a *unique* deletion sequence corresponding to every Prüfer code. Therefore there are n^{n-2} possible deletion sequences.

Each deletion sequence tells us every single edge of the tree it came from, and every tree has a uniquely-defined deletion sequence. Therefore there are n^{n-2} possible labeled n-vertex trees. □

3.2 Information hidden in Prüfer codes

The Prüfer code of a tree actually contains a surprising amount of information about the tree.

Proposition 3.2. *In the Prüfer code of a tree where $\deg(v_i) = k$, the number i appears $k-1$ times.*

Proof. When we fill in the blanks in the sequence

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

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

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

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

But this sequence is just directly telling us what the edges of the tree are: if a number i appears k times in this sequence, that means that v_i appears in k edges. Therefore $\deg(v_i) = k$: one more than the number of times i appears in the Prüfer code. □

For instance, the Prüfer code 44622 from our previous example can only be the Prüfer code of a tree with vertices $v_1, v_2, v_3, v_4, v_5, v_6, v_7$ in which $\deg(v_2) = 3$, $\deg(v_4) = 3$, $\deg(v_6) = 2$, and $\deg(v_1) = \deg(v_3) = \deg(v_5) = \deg(v_7) = 1$.

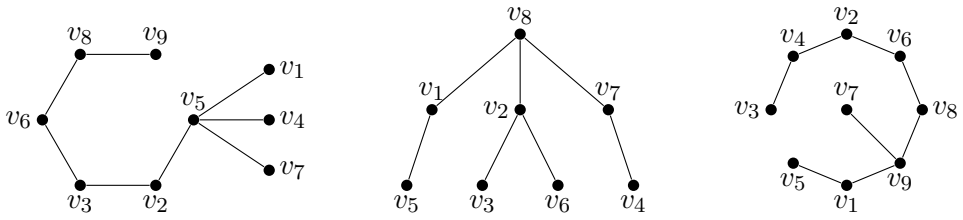
Many counting problems about labeled can be solved very quickly with the use of Proposition 3.2. For example:

Corollary 3.3. *There are $(n-1)^{n-2}$ trees with vertex set $\{v_1, v_2, \ldots, v_n\}$ in which vertex v_1 is a leaf.*

Proof. By Proposition 3.2, vertex v_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: there are $n-1$ possibilities for each value in the code, and $n-2$ values. Therefore there are $(n-1)^{n-2}$ such trees. □

4 Practice problems

1. Find the trees with the following Prüfer codes:
   (a) 11111.
   (b) 123456.
   (c) 3141592.
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 $\{v_1, v_2, v_3, v_4\}$. (Prüfer codes are not very useful here.)
4. What does the Prüfer code look like for a tree with vertices $\{v_1, v_2, \ldots, v_n\}$ when...
   (a) ...that tree is a star (it is isomorphic to $K_{1,n-1}$)? How many such trees are there?
   (b) ...that tree is a path (it is isomorphic to P_n)? How many such trees are there?
5. Use Prüfer codes and Proposition 3.2 to count:
   (a) The number of trees with vertex set $\{v_1, v_2, \ldots, v_n\}$ in which v_1 has degree 3.
   (b) The number of trees with vertex set $\{v_1, v_2, \ldots, v_n\}$ in which v_1 has degree $n-2$.
   (c) The number of trees with vertex set $\{v_1, v_2, v_3, v_4, v_5, v_6\}$ where $\deg(v_1) = \deg(v_2) = 3$ and $\deg(v_3) = \deg(v_4) = \deg(v_5) = \deg(v_6) = 1$.

   For parts (b) and (c), think about how you would count them without using Prüfer codes.
6. Use Corollary 3.3 to find the average number of leaves in an n-vertex labeled tree.
7. Prove that if a tree has maximum degree d, then it has at least d leaves:
   (a) Using Prüfer codes and Proposition 3.2.
   (b) Using ideas from the previous lecture.
8. Each possible edge v_iv_j is contained in $f(n)$ of the n^{n-2} trees with vertex set $\{v_1, v_2, \ldots, v_n\}$. By symmetry, $f(n)$ is the same for any edge v_iv_j.
   (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.