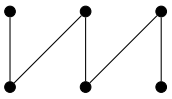
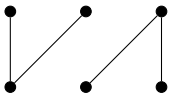
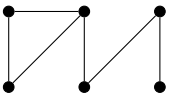
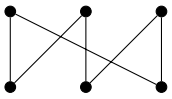
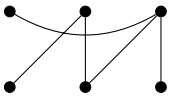
Math 365 – Monday 5/6/19 – Trees (11.1 & 11.4)

Exercise 58. (a) Which of the following graphs are trees? Which are forests?

(i)  (ii)  (iii)  (iv)  (v) 

(b) What is the smallest n for which there is a tree on n vertices that is not a path?

(c) What is the largest number of leaves a tree on n vertices can have, for $n \geq 3$?

(d) How many isomorphism classes are there of trees on 4 vertices? Draw them.

(e) How many isomorphism classes are there of forests on 4 vertices? Draw them.

(f) How many isomorphism classes are there of trees on 5 vertices? Draw them.

(g) How many isomorphism classes are there of forests on 5 vertices? Draw them.

(h) For which values of n is K_n a tree? For which values of m, n is $K_{m,n}$ a tree?

(i) If T is a tree, what are $\kappa(T)$, $\lambda(T)$, and $\omega(T)$? What can you say about $\alpha(T)$?

(j) Which trees have Euler trails? Which trees have paths that visit every vertex ("Hamilton paths")?

(k) What does the Handshake theorem tell you about the degrees sequence of a tree?

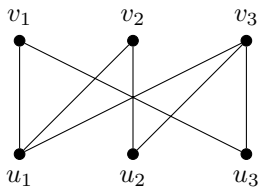
(l) Explain why every tree is 2-colorable (and therefore bipartite). [Hint: describe a process for 2-coloring a tree.]

(m) Your answer to (e) should have been 2. Now calculate the number of 0, 1, 2, 3, and 4-colorings of a labeled representative of each tree to verify that the chromatic polynomial is the same across all trees on 4 vertices.

(n) Explain why a graph is a tree if and only if it is connected and has $|V| - 1$ edges.
[Hint: You already know one direction, the "if G is a tree, then…" direction. Now suppose G is not a tree. Then either it's not connected, or it has a cycle (say on m vertices). If it has a cycle, there's an induced subgraph that is a cycle. Start from there, and build G up one vertex at a time. What's the minimum number of edges you have to accumulate?]

Exercise 59. (a) How many spanning trees does C_n have for $n = 3, 4, 5,?$

(b) Use the recurrence relation $t(G) = t(G - e) + t(G/e)$ to count the number of spanning trees of



Remember to keep multiple edges!!

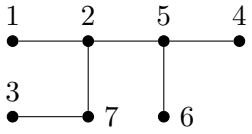
(c) How many spanning trees does W_n have for $n = 3, 4, 5,?$

(d) How many spanning trees does K_n have for $n = 4, 5?$

(e) Explain why a tree has exactly one spanning tree.

(f) Is it true that every maximal path of G is also a maximal path of some spanning tree? Do some examples, and explain why or why not. (Careful: Recall that "maximal path" and "maximal length path" mean different things.)

Exercise 60. (a) What is the Prüfer code for the following labeled tree?



Check your answer by reversing the process and building the tree from the code.

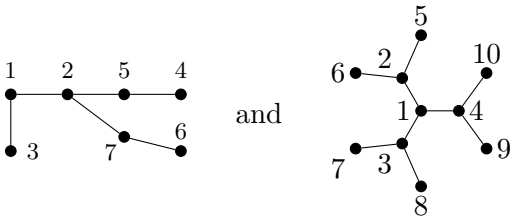
(b) Draw the tree whose Prüfer code is $2, 2, 5, 3, 6$. Check your answer by calculating the Prüfer code that goes with your tree.

(c) Draw a labeled K_3 (labeled with $1, 2, 3$), and list all the spanning trees, and the corresponding Prüfer code. Verify that there is a bijection between the labeled trees on 3 vertices and the length-1 Prüfer codes.

(d) How many spanning trees does K_7 have?

(e) How many labeled trees are there on 14 vertices?

(f) In class we computed that the codes for



are $1, 2, 5, 2, 7$ and $2, 2, 1, 3, 3, 1, 4, 4$, respectively. Compare this to the degrees each of the (labeled) vertices in corresponding trees. Add to that data your computation from parts (a), (b), and (c), and collect all this into a table of the form:

Püfer code	d_1	d_2	$\cdots$

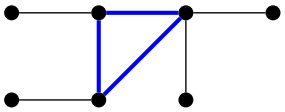
(where d_1 is the degree of vertex 1, d_2 is the degree of vertex 2, and so on). Now make a hypothesis about a correspondence between some properties of the Prüfer code and degrees of a labeled tree. Use your hypothesis to explain why the number of labeled trees where vertex i has degree d_i is

$$\frac{(n-2)!}{(d_1-1)!\cdots(d_n-1)!}$$

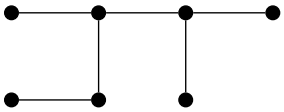
(think back to counting techniques of chapter 6!). More generally, what can you say about the correspondence between Prüfer codes and degree sequences?

We say a graph is acyclic if it doesn't have any cycles.

Has a cycle:

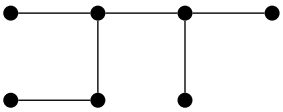


Acyclic:

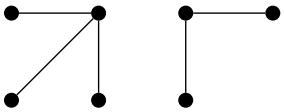


A tree is a connected acyclic graph. A forrest a collection of trees (i.e. a not necessarily connected acyclic graph).

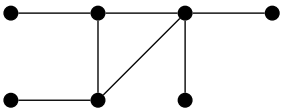
Tree and forrest:



Forrest but not tree :



Not tree nor forrest:



Note that the connected components of a forrest are trees.
A leaf is a vertex of degree 1.

Lemma

Every tree with at least two vertices has at least two leaves.

A tree is a connected acyclic graph.

Theorem

A tree with n vertices has exactly $n-1$ edges.

Prove by induction on the number of vertices.

Since every connected component of a forrest is a tree, we get the following as a corollary.

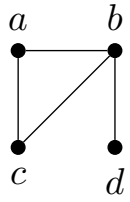
Corollary

A forrest with k connected components has exactly $|V|-k$ edges.

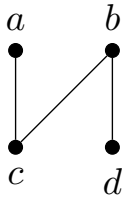
You try: Exercise 58.

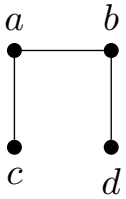
Spanning trees

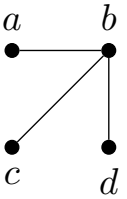
A spanning tree for a connected graph G is a subgraph of G with the same vertex set, but that is itself a tree. For example, the graph

$G =$ 

has exactly three spanning trees:

$T_1 =$ 

, $T_2 =$ 

, and $T_3 =$ 

.

(G had once cycle. Deleting one edge from that cycle leaves you with a tree.)

Counting spanning trees

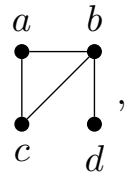
For a connected graph G, let $t(G)$ be the number of spanning trees in G (also a graph invariant).

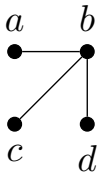
How to count $t(G)$: Notice that for any fixed edge, you can split the spanning trees into two categories: (1) those that do not contain e and (2) those that do.

Case 1: Every spanning tree of G that doesn't contain e is also a spanning tree of $G - e$, so

$|\{$ spanning trees of G not containing edge $e\ \}| = t(G - e)$.

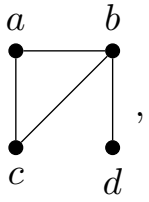
For example, in G from before, fix $e = a{-}c$ in

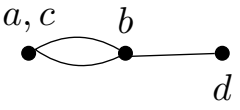
$G =$ 

, the only spanning tree not containing e is 

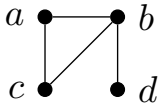
which is the only spanning tree of $G - e$ (which *is* the tree).

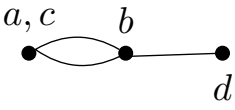
For a connected graph G, let $t(G)$ be the number of spanning trees in G (also a graph invariant).
How to count $t(G)$: For any edge e, break into cases: (1) those that do not contain e and (2) those that do.
Case 1:
$|\{$ spanning trees of G not containing edge $e\ \}| = t(G - e)$.
Case 2: count trees containing e.
Recall G/e be the graph gotten by glueing the endpoints of e and deleting e. For example, if e is the edge joining a and c in

$G =$ 

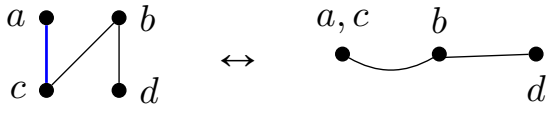
, then $G/e =$ 

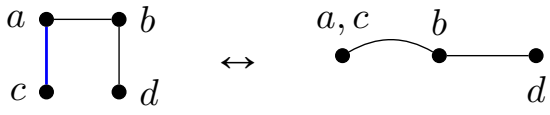
Case 2: count trees containing e.
Recall G/e be the graph gotten by glueing the endpoints of e and deleting e. For example, if e is the edge joining a and c in

$G =$ 

, then $G/e =$ 

And the spanning trees of G that contain e are in bijection with the spanning trees of G/e:





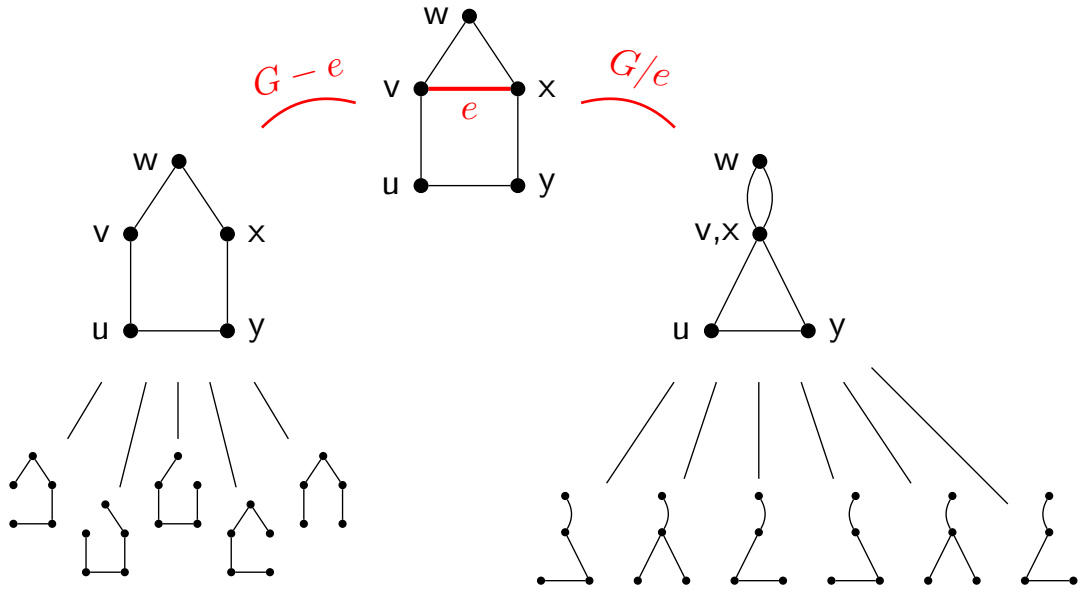
In general,

$|\{$ spanning trees of G containing edge $e\ \}| = t(G/e)$.

For a connected graph G, let $t(G)$ be the number of spanning trees in G (also a graph invariant).
How to count $t(G)$: For any edge e, break into cases: (1) those that do not contain e and (2) those that do.
Case 1:
$|\{$ spanning trees of G not containing edge e $\}| = t(G - e)$.
Case 2: $|\{$ spanning trees of G containing edge e $\}| = t(G/e)$.

So

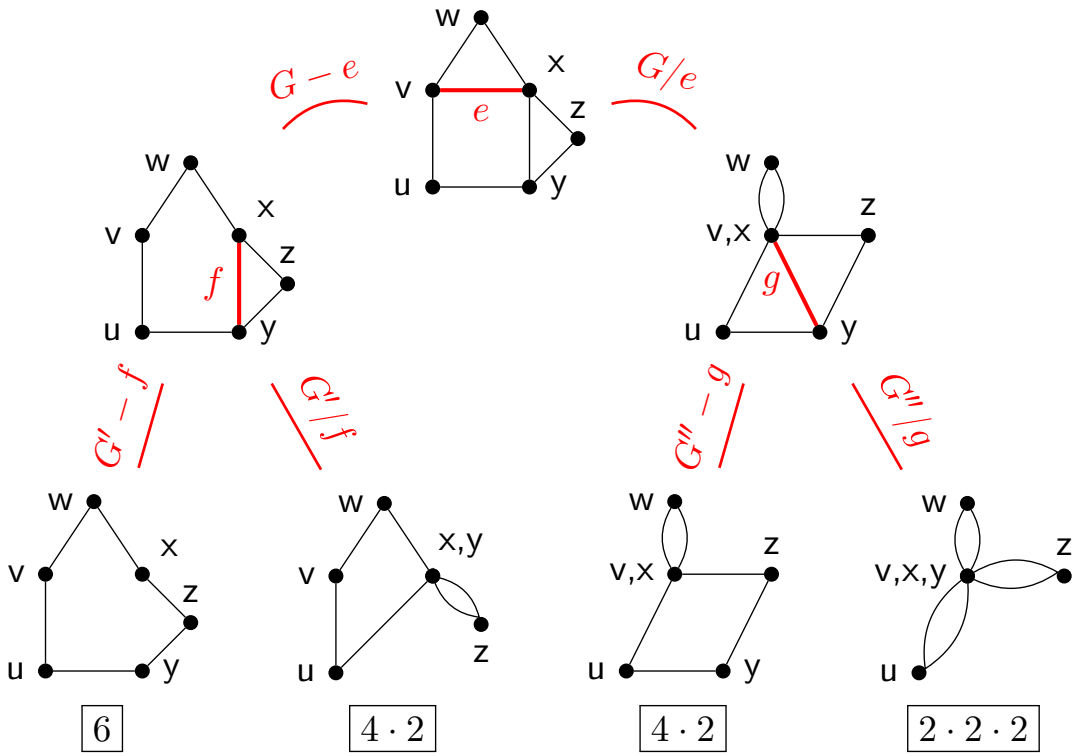
$$t(G) = t(G - e) + t(G/e).$$



removing any edge of a cycle produces a spanning tree: 5 of these

for each cycle, removing exactly one edge produces a spanning tree: $2 \cdot 3$ of these

Total: $5 + 2 \cdot 3 = 11$ spanning trees

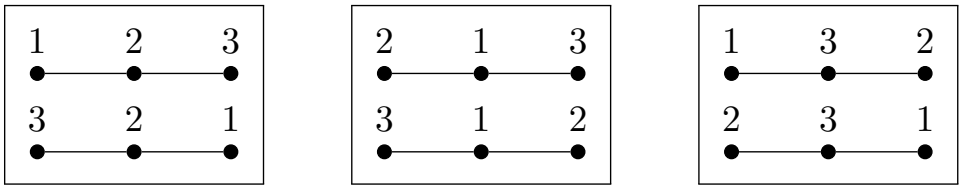


Total: $6 + 4 \cdot 2 + 4 \cdot 2 + 2 \cdot 2 \cdot 2 = 30$ spanning trees

Counting trees

The goal is to count the number of trees with n vertices labeled $\{1, 2, 3, \ldots, n\}$. For example, up to isomorphism, there is exactly one tree with three vertices:

If I naively try to label the three vertices with $\{1, 2, 3\}$, I would get 6 results:

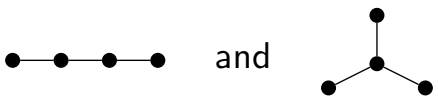


But actually, the first two are just drawings of the same tree; so are the second two; so are the last two!

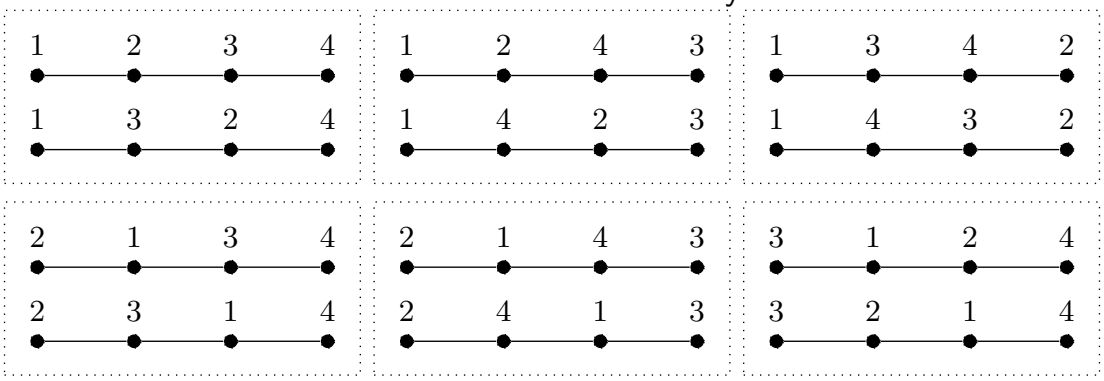
So there are $\boxed{3}$ labeled trees on 3 vertices.

Goal: Count # trees with n vertices labeled $\{1, \ldots, n\}$.

Example: Count the number of labeled trees with 4 vertices.
For example, up to isomorphism, there are exactly two trees with four vertices:



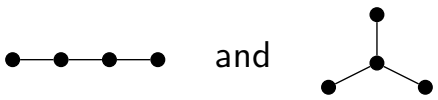
For the path: choose the outer vertices – $\binom{4}{2}$ ways, and then choose the order of the inner vertices – 2 ways.



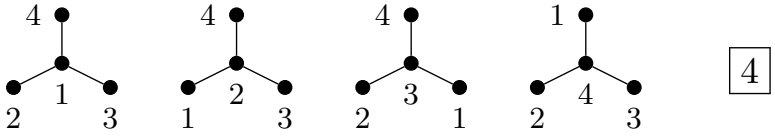
So there are $\binom{4}{2} \cdot 2 = 6 \cdot 2 = \boxed{12}$ of these.

Goal: Count # trees with n vertices labeled $\{1, \ldots, n\}$.

Example: Count the number of labeled trees with 4 vertices.
For example, up to isomorphism, there are exactly two trees with four vertices:

 and

For the path: choose the outer vertices – $\binom{4}{2}$ ways, and then choose the order of the inner vertices – 2 ways. So there are $\binom{4}{2} \cdot 2 = 6 \cdot 2 = \boxed{12}$ of these.
For the star, choosing the label for the middle vertex determines the tree:



$\boxed{4}$

Total: $12 + 4 = \boxed{16}$.

Approach: find a bijection with something that's easier to count.

$$\{ \text{ labeled trees with } n \text{ vertices } \}$$
$$\leftrightarrow$$
$$\{ \text{ sequence of length } n-2 \text{ from } \{1, \ldots, n\} \}$$

The associated sequence is called the tree's Prüfer code.

Built as follows:

Prüfer code from tree:
1. Remove the lowest leaf possible and record its neighbor.
2. Iterate until there are exactly two leaves left.

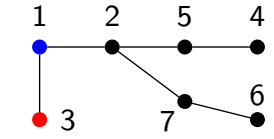
Your code should have $n - 2$ numbers.

Prüfer code from tree:

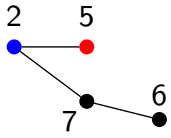
1. Remove the lowest leaf possible and record its neighbor.
2. Iterate until there are exactly two leaves left.

Your code should have $n-2$ numbers.

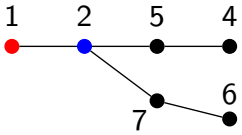
Example:



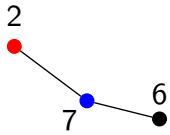
code: $\varnothing$



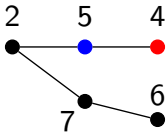
code: $1, 2, 5$



code: 1



code: $1, 2, 5, 2$



code: $1, 2$

code: $1, 2, 5, 2, 7$

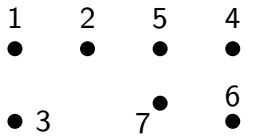
Done! Prüfer code: $\boxed{1, 2, 5, 2, 7}$.

Reversing this process:

Tree from Prüfer code:

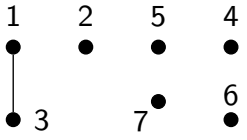
1. draw a bar ($|$) at the end of your code of length $n-2$, and draw n vertices, labeled from 1 to n.
2. Let a be the first number in the code, and b be the smallest missing number. (i) draw an edge from a to b, (ii) delete a, and (iii) put b at the end (after the $|$).
3. Recurse until you've cycled the bar to the front. Then draw and edge between the two numbers that are missing from your code.

Example: Take the code $1, 2, 5, 2, 7$.



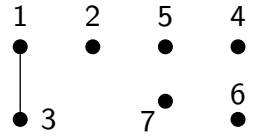
code: $1, 2, 5, 2, 7|$

missing: $3, 4, 6$

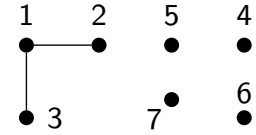


code: $2, 5, 2, 7|3$

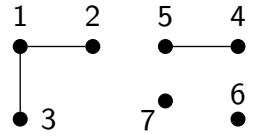
missing: $1, 4, 6$



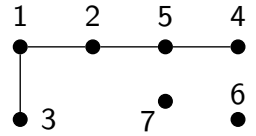
code: $2, 5, 2, 7|3$
missing: $1, 4, 6$



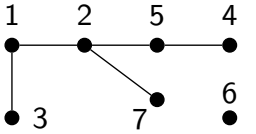
code: $5, 2, 7|3, 1$
missing: $4, 6$



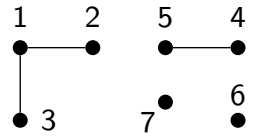
code: $2, 7|3, 1, 4$
missing: 5, 6



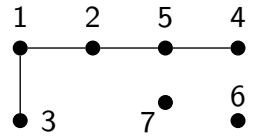
code: $7|3, 1, 4, 5$
missing: $2, 6$



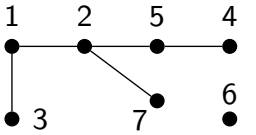
code: $|3, 1, 4, 5, 2$
missing: $6, 7$



code: $2, 7|3, 1, 4$
missing: 5, 6

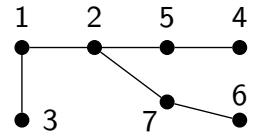


code: $7|3, 1, 4, 5$
missing: $2, 6$



code: $|3, 1, 4, 5, 2$
missing: $6, 7$

3. Recurse until you've cycled the bar to the front. Then draw and edge between the two numbers that are missing from your code.

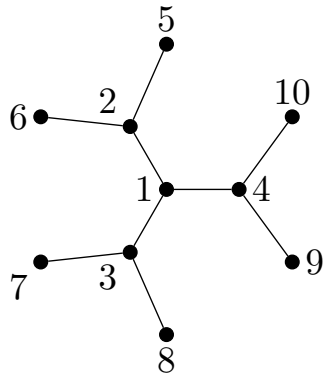


Done!
This is the tree!

Same tree as before!

You try:

1. Calculate the Prüfer code for the following tree



   and verify your answer by then computing the tree that comes from that code, and checking that they match.

2. Compute the tree that corresponds to the Prüfer code that corresponds to the sequence $1, 5, 4, 4, 3$, and verify your answer by by then computing the code that comes from that tree, and checking that they match.

These two processes are precisely inverses of each other!
Therefore, for each n, there is a bijection

$$\{ \text{ labeled trees with } n \text{ vertices } \}$$
$$\leftrightarrow$$
$$\{ \text{ sequences of length } n-2 \text{ from } \{1,\ldots,n\} \}$$

via Prüfer codes.

Theorem (Cayley's formula)

There are n^{n-2} labeled trees on n vertices.

Proof: There are $\underbrace{n \cdot n \cdots n}_{n-2}$ sequences of length $n-2$ from $\{1,\ldots,n\}$. □

Further:: very labeled tree with n vertices is a spanning tree of (a labeled) K_n, and vice versa.

Corollary

There are n^{n-2} spanning trees in K_n.