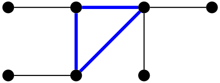
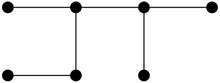
We say a graph is acyclic if it doesn't have any cycles.

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

Has a cycle:

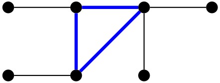


Acyclic:

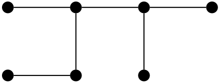


We say a graph is acyclic if it doesn't have any cycles.

Has a cycle:



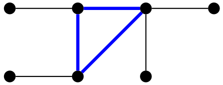
Acyclic:



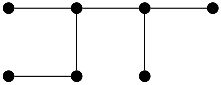
A tree is a connected acyclic graph.

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

Has a cycle:



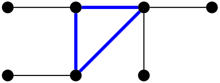
Acyclic:



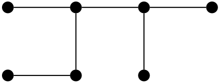
A tree is a connected acyclic graph. A forrest a collection of trees (i.e. a not necessarily connected acyclic graph).

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

Has a cycle:

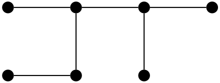


Acyclic:

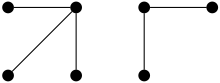


A tree is a connected acyclic graph. A forrest a collection of trees (i.e. a not necessarily connected acyclic graph).

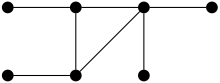
Tree and forrest:



Forrest but not tree :

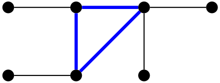


Not tree nor forrest:

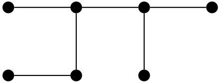


We say a graph is acyclic if it doesn't have any cycles.

Has a cycle:

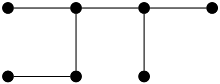


Acyclic:

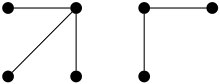


A tree is a connected acyclic graph. A forrest a collection of trees (i.e. a not necessarily connected acyclic graph).

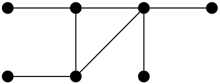
Tree and forrest:



Forrest but not tree :



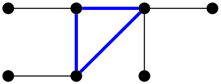
Not tree nor forrest:



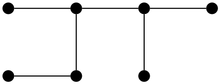
Note that the connected components of a forrest are trees.

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

Has a cycle:

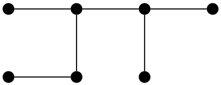


Acyclic:

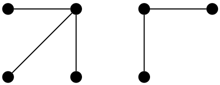


A tree is a connected acyclic graph. A forrest a collection of trees (i.e. a not necessarily connected acyclic graph).

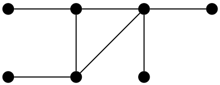
Tree and forrest:



Forrest but not tree :



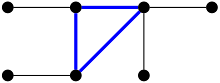
Not tree nor forrest:



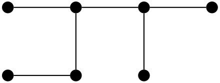
Note that the connected components of a forrest are trees.
A leaf is a vertex of degree 1.

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

Has a cycle:

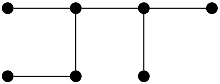


Acyclic:

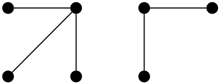


A tree is a connected acyclic graph. A forrest a collection of trees (i.e. a not necessarily connected acyclic graph).

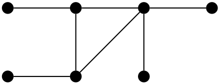
Tree and forrest:



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.

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.

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.

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.

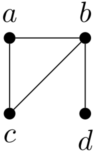
You try: Exercise 58.

Spanning trees

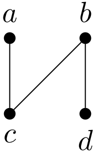
A spanning tree for a connected graph G is a subgraph of G with the same vertex set, but that is itself a tree.

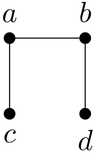
Spanning trees

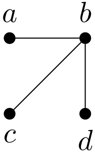
A spanning tree for a connected graph G is a subgraph of G with the same vertex set, but that is itself a tree. For example, the graph

$G =$ 

has exactly three spanning trees:

$T_1 =$ 

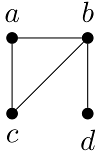
, $T_2 =$ 

, and $T_3 =$ 

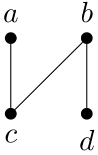
.

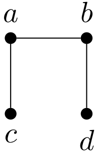
Spanning trees

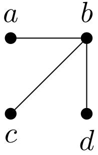
A spanning tree for a connected graph G is a subgraph of G with the same vertex set, but that is itself a tree. For example, the graph

$G =$ 

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).

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 $t(G)$:

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.

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 $G - e$

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

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

Counting spanning trees

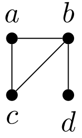
For a connected graph G, let $t(G)$ be the number of spanning trees in G (also a graph invariant).

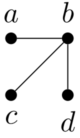
How to count $t(G)$: Notice that for any fixed edge, you can split the spanning trees into two categories: (1) those that do not contain e and (2) those that do.

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

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

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.

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

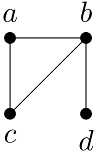
How to count $t(G)$: For any edge e, break into cases: (1) those that do not contain e and (2) those that do.

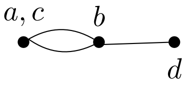
Case 1:

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

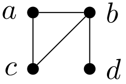
Case 2: count trees containing e.

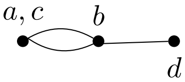
Recall G/e be the graph gotten by glueing the endpoints of e and deleting e. For example, if e is the edge joining a and c in

$G =$ 

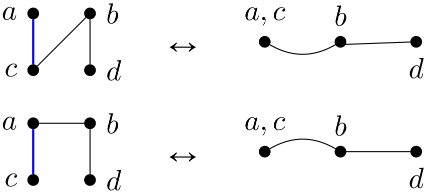
, then $G/e =$ 

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

$G =$ 

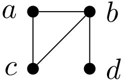
, then $G/e =$ 

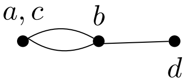
And the spanning trees of G that contain e are in bijection with the spanning trees of G/e:



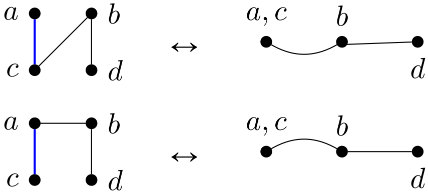
Case 2: count trees containing e.

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:



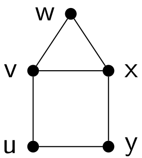
In general,

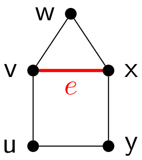
$$|\{\text{ spanning trees of } G \text{ containing edge } e\ \}| = t(G/e).$$

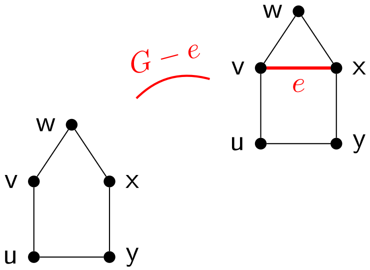
For a connected graph G, let $t(G)$ be the number of spanning trees in G (also a graph invariant).
How to count $t(G)$: For any edge e, break into cases: (1) those that do not contain e and (2) those that do.
Case 1:
$|\{$ spanning trees of G not containing edge e $\}| = t(G - e)$.
Case 2: $|\{$ spanning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

So

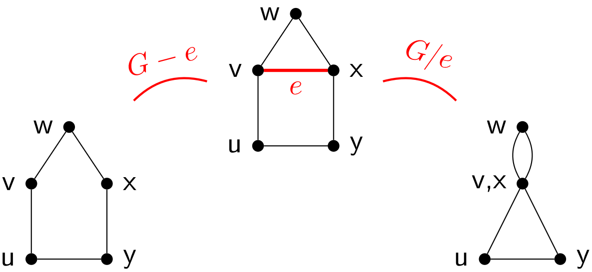
$$t(G) = t(G - e) + t(G/e).$$

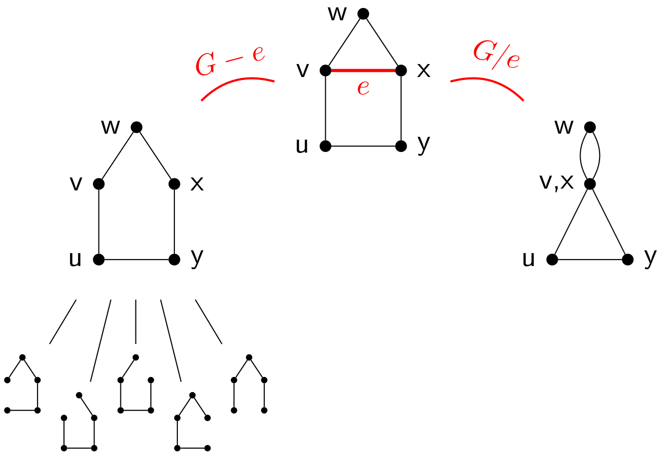

w
v
x
u
y


w
v
x
e
u
y

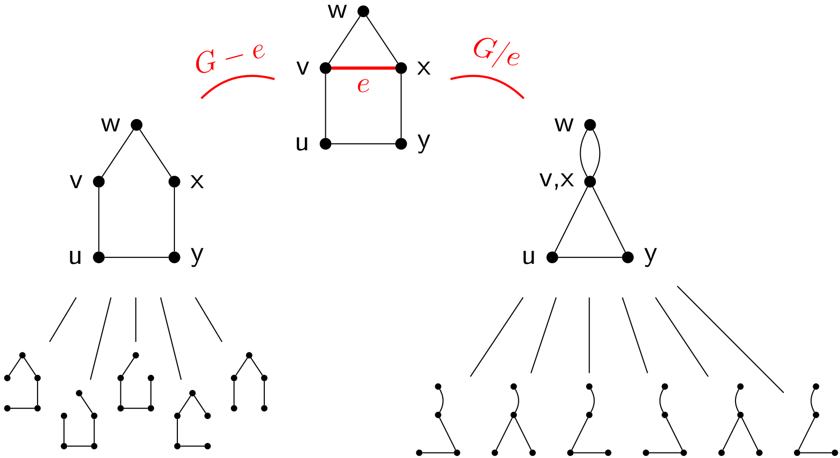


w
v
x
u
y
$G - e$
w
v
x
e
u
y


G − e
G/e
w
v
x
e
u
y
w
v
x
u
y
w
v,x
u
y

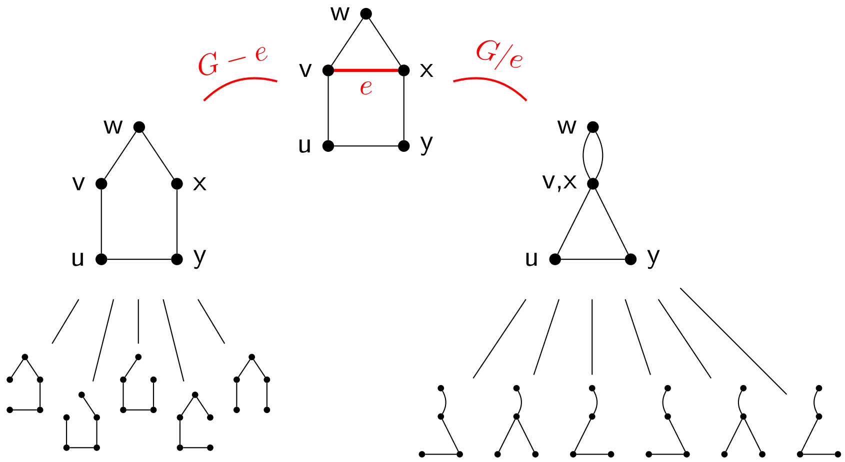


removing any edge of a cycle
produces a spanning tree:
5 of these



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

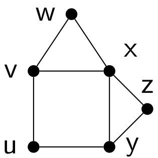
for each cycle, removing exactly one
edge produces a spanning tree:
$2 \cdot 3$ of these

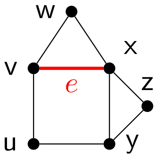


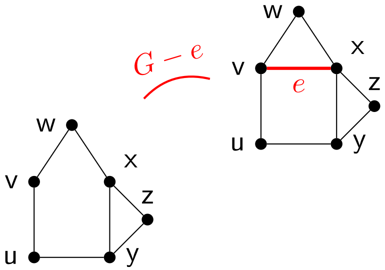
removing any edge of a cycle
produces a spanning tree:
5 of these

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

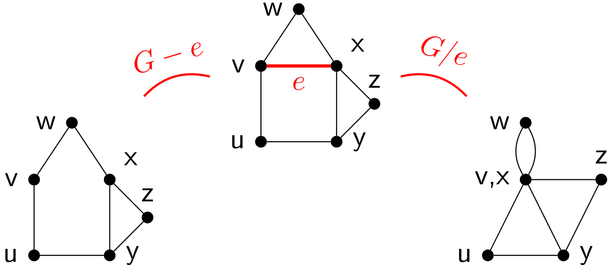
Total: $5 + 2 \cdot 3 = 11$ spanning trees

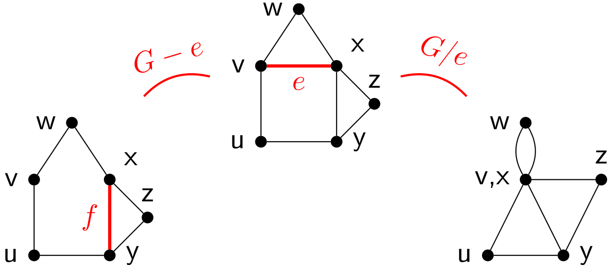

w
x
v
z
u
y


w
x
v
z
e
u
y

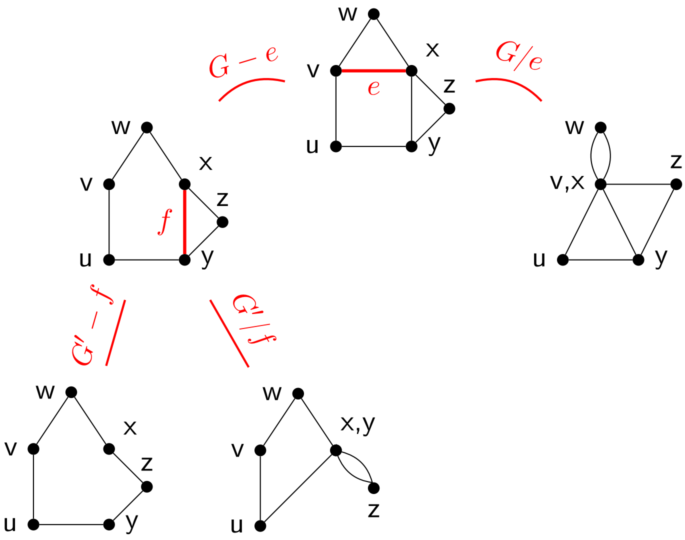


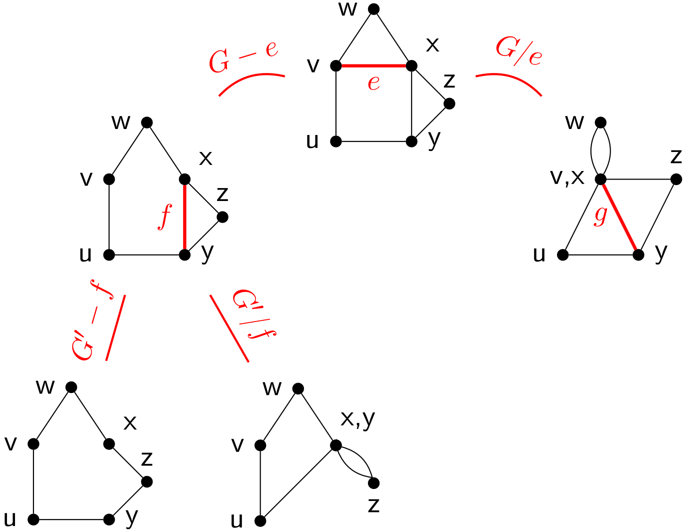
G − e
w
x
v
e
z
u
y
w
x
v
z
u
y

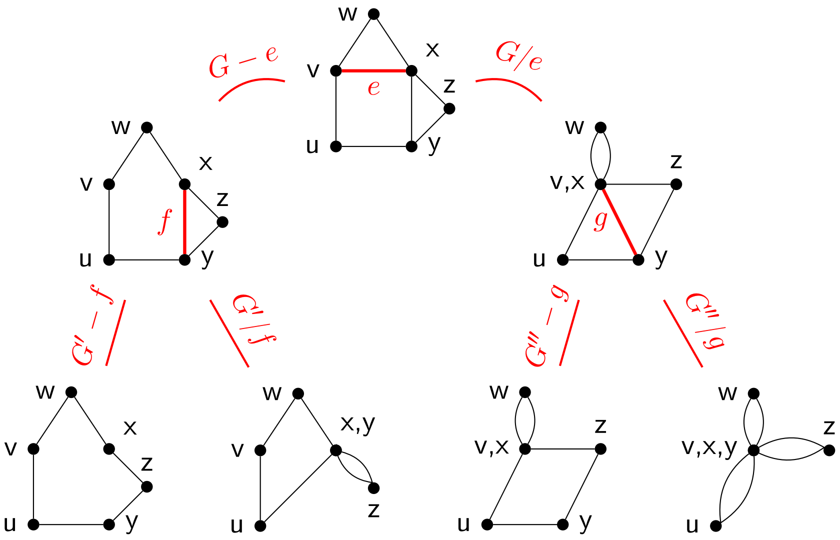

w
x
v
e
z
u
y
$G - e$
G/e
w
v
x
z
u
y
w
z
v,x
u
y

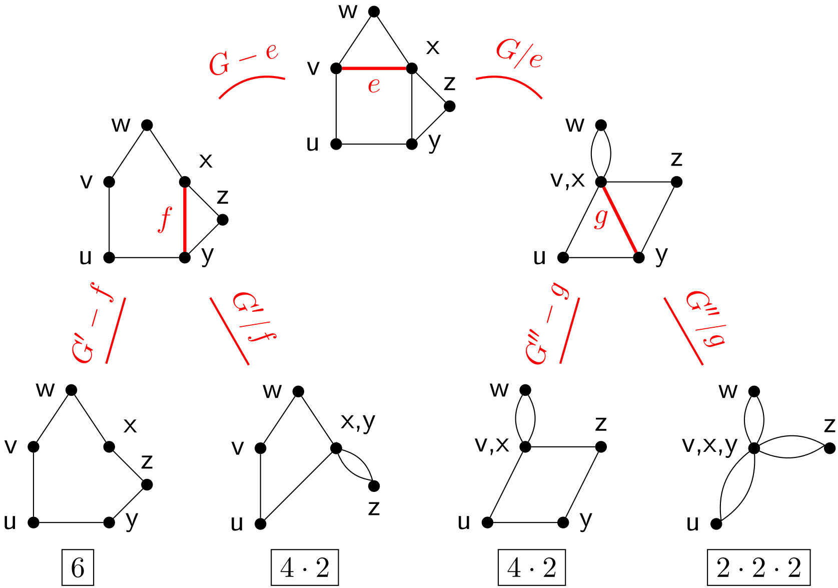


w
x
v
e
z
u
y
$G - e$
G/e
w
v
x
z
f
u
y
w
z
v,x
u
y

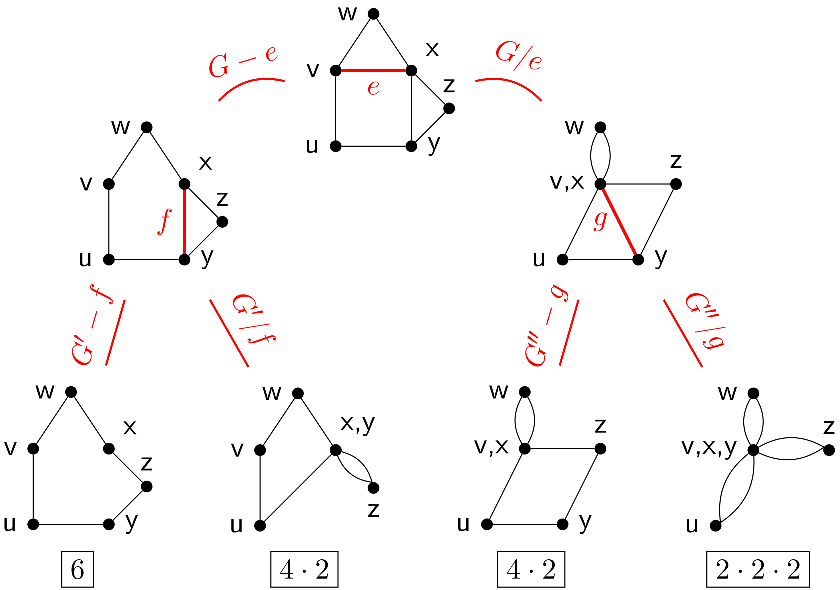

w
x
v
e
z
u
y
$G - e$
G/e
w
v
x
z
f
u
y
w
v,x
z
u
y
$G' - f$
G'/f
w
x
v
z
u
y
w
x,y
v
z
u

$G - e$
G/e
w
v
x
z
u
y
e
w
v
x
z
u
y
f
w
v,x
z
u
y
g
$G' - f$
G'/f
w
v
x
z
u
y
w
v
x,y
z
u


w
v
x
z
u
y
e
G − e
G/e
w
v
x
z
f
u
y
w
v,x
z
g
u
y
G′ − f
G′/f
G″ − g
G″/g
w
v
x
z
u
y
w
v
x,y
u
z
w
v,x
z
u
y
w
v,x,y
z
u



w
v
x
z
u
y
e
$G - e$
G/e
f
g
v,x
$G' - f$
G'/f
$G'' - g$
G''/g
x,y
v,x,y
6
$4 \cdot 2$
$4 \cdot 2$
$2 \cdot 2 \cdot 2$



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

Counting trees

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

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

for any edge e.

Counting trees

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

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

for any edge e.

Counting trees in general: The goal is to count the number of trees with n vertices labeled $\{1, 2, 3, \ldots, n\}$.

Counting trees

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

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

for any edge e.

Counting trees in general: 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:

Counting trees

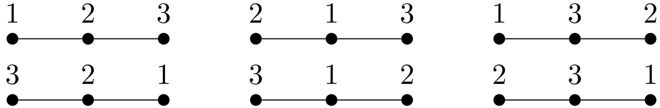
For a connected graph G, let $t(G)$ be the number of spanning trees in G (also a graph invariant). We have the recursive formula

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

for any edge e.

Counting trees in general: 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:

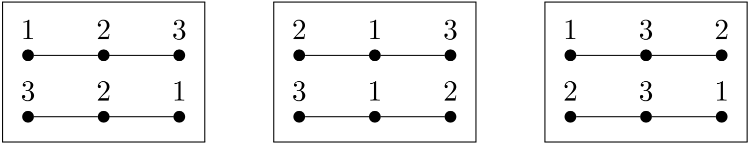
If I naively try to label the three vertices with $\{1, 2, 3\}$, I would get 6 results:



Counting trees

The goal is to count the number of trees with n vertices labeled $\{1, 2, 3, \dots, n\}$. 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:



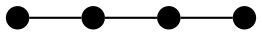
But actually, the first two are just drawings of the same tree; so are the second two; so are the last two!

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

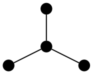
Goal: Count # trees with n vertices labeled $\{1, \ldots, n\}$.

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:

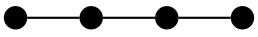
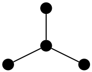


and

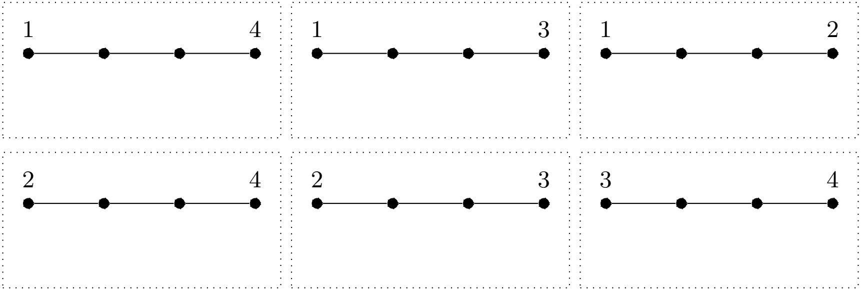


Goal: Count # trees with n vertices labeled $\{1, \ldots, n\}$.

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

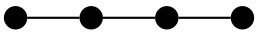

and


For the path: choose the outer vertices – $\binom{4}{2}$ ways

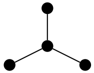


Goal: Count # trees with n vertices labeled $\{1, \ldots, n\}$.

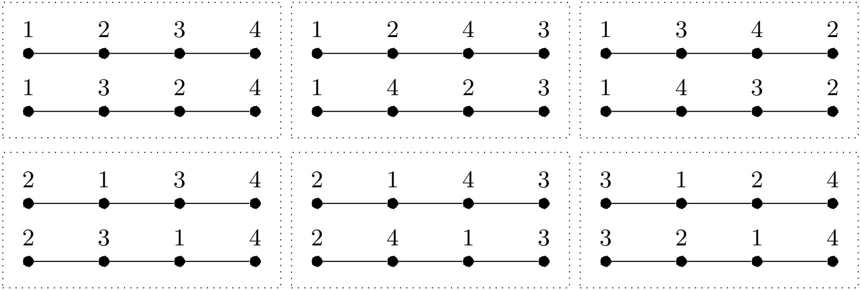
Example: Count the number of labeled trees with 4 vertices.
For example, up to isomorphism, there are exactly two trees with four vertices:



and

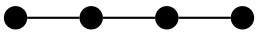
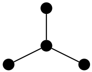


For the path: choose the outer vertices – $\binom{4}{2}$ ways, and then choose the order of the inner vertices – 2 ways.

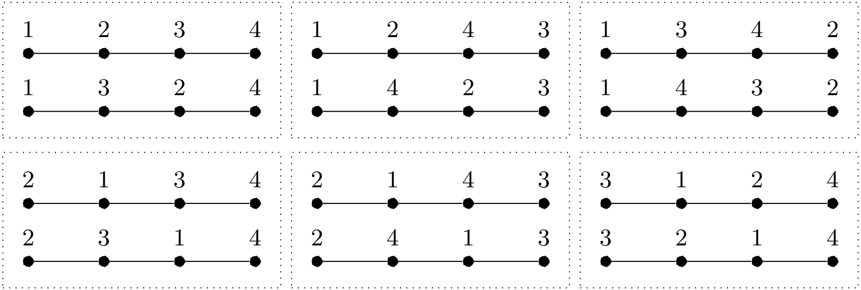


Goal: Count # trees with n vertices labeled $\{1, \ldots, n\}$.

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.

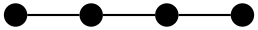


So there are $\binom{4}{2} \cdot 2 = 6 \cdot 2 = \boxed{12}$ of these.

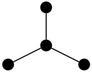
Goal: Count # trees with n vertices labeled $\{1, \ldots, n\}$.

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:

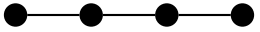
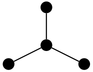


and

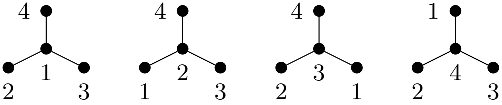


Goal: Count # trees with n vertices labeled $\{1, \dots, n\}$.

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

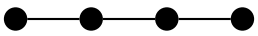
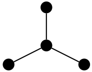

and


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:

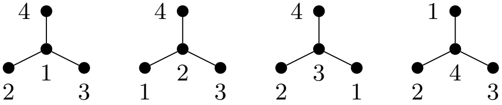


Goal: Count # trees with n vertices labeled $\{1, \ldots, n\}$.

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

 and 

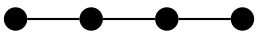
For the path: choose the outer vertices – $\binom{4}{2}$ ways, and then choose the order of the inner vertices – 2 ways. So there are $\binom{4}{2} \cdot 2 = 6 \cdot 2 = \boxed{12}$ of these.
For the star, choosing the label for the middle vertex determines the tree:



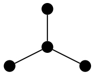
$\boxed{4}$

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

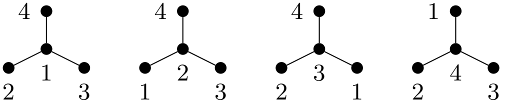
Example: Count the number of labeled trees with 4 vertices.
For example, up to isomorphism, there are exactly two trees with four vertices:



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.

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

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

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.

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:

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.

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.

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.

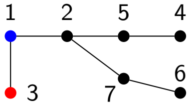
Your code should have $n-2$ numbers.

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:



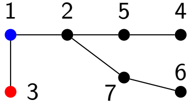
code: $\varnothing$

Prüfer code from tree:

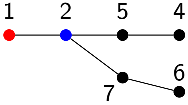
1. Remove the lowest leaf possible and record its neighbor.
2. Iterate until there are exactly two leaves left.

Your code should have $n-2$ numbers.

Example:



code: $\varnothing$



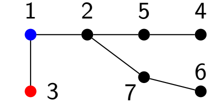
code: 1

Prüfer code from tree:

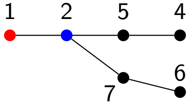
1. Remove the lowest leaf possible and record its neighbor.
2. Iterate until there are exactly two leaves left.

Your code should have $n-2$ numbers.

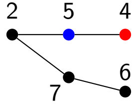
Example:



code: $\varnothing$



code: 1



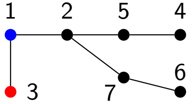
code: $1, 2$

Prüfer code from tree:

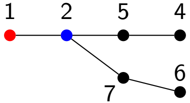
1. Remove the lowest leaf possible and record its neighbor.
2. Iterate until there are exactly two leaves left.

Your code should have $n - 2$ numbers.

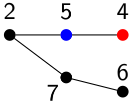
Example:



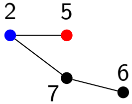
code: $\varnothing$



code: 1



code: $1, 2$



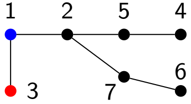
code: $1, 2, 5$

Prüfer code from tree:

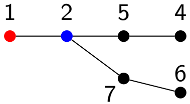
1. Remove the lowest leaf possible and record its neighbor.
2. Iterate until there are exactly two leaves left.

Your code should have $n-2$ numbers.

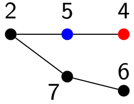
Example:



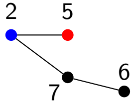
code: $\varnothing$



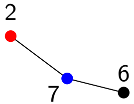
code: 1



code: $1, 2$



code: $1, 2, 5$



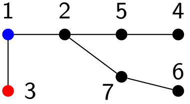
code: $1, 2, 5, 2$

Prüfer code from tree:

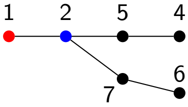
1. Remove the lowest leaf possible and record its neighbor.
2. Iterate until there are exactly two leaves left.

Your code should have $n-2$ numbers.

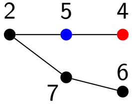
Example:



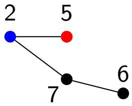
code: $\varnothing$



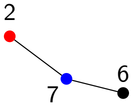
code: 1



code: $1, 2$



code: $1, 2, 5$



code: $1, 2, 5, 2$

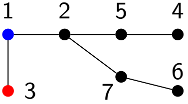
code: $1, 2, 5, 2, 7$

Prüfer code from tree:

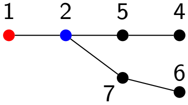
1. Remove the lowest leaf possible and record its neighbor.
2. Iterate until there are exactly two leaves left.

Your code should have $n-2$ numbers.

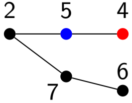
Example:



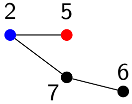
code: $\varnothing$



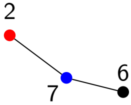
code: 1



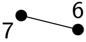
code: $1, 2$



code: $1, 2, 5$



code: $1, 2, 5, 2$



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.

Reversing this process:

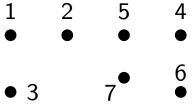
Tree from Prüfer code:

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: $3, 4, 6$

Reversing this process:

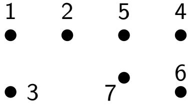
Tree from Prüfer code:

1. draw a bar (|) at the end of your code of length $n-2$, and draw n vertices, labeled from 1 to n.

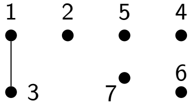
2. Let a be the first number in the code, and b be the smallest missing number. (i) draw an edge from a to b, (ii) delete a, and (iii) put b at the end (after the $|$).

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 $1, 2, 5, 2, 7$.



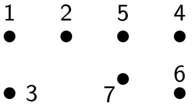
code: $1, 2, 5, 2, 7|$
missing: $3, 4, 6$



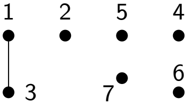
code: $2, 5, 2, 7|3$
missing: $1, 4, 6$

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.

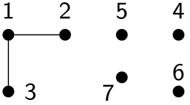
Example: Take the code $1, 2, 5, 2, 7$.



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



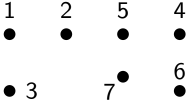
code: $2, 5, 2, 7|3$
missing: $1, 4, 6$



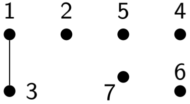
code: $5, 2, 7|3, 1$
missing: $4, 6$

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.

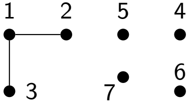
Example: Take the code $1, 2, 5, 2, 7$.



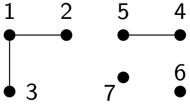
code: $1, 2, 5, 2, 7|$
missing: $3, 4, 6$



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

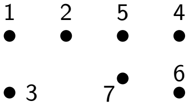


code: $5, 2, 7|3, 1$
missing: $4, 6$

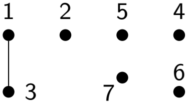


code: $2, 7|3, 1, 4$
missing: 5, 6

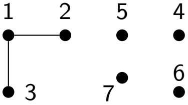
Example: Take the code $1, 2, 5, 2, 7$.



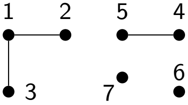
code: $1, 2, 5, 2, 7|$
missing: $3, 4, 6$



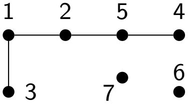
code: $2, 5, 2, 7|3$
missing: $1, 4, 6$



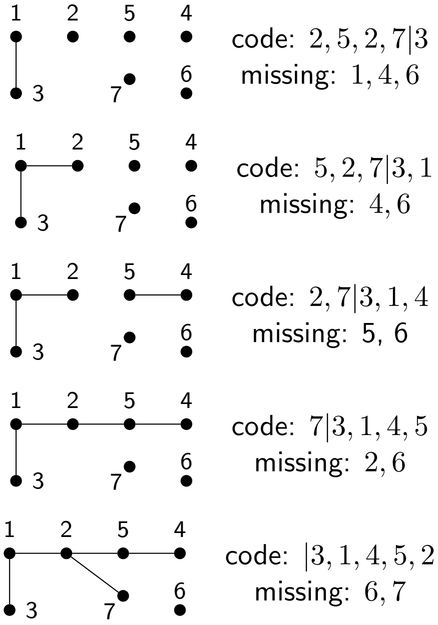
code: $5, 2, 7|3, 1$
missing: $4, 6$



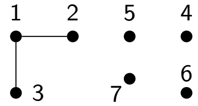
code: $2, 7|3, 1, 4$
missing: 5, 6



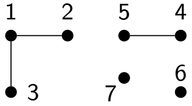
code: $7|3, 1, 4, 5$
missing: $2, 6$



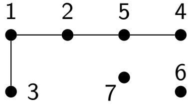
1 2 5 4 3 7 6
code: $2, 5, 2, 7|3$
missing: $1, 4, 6$
1 2 5 4 3 7 6
code: $5, 2, 7|3, 1$
missing: $4, 6$
1 2 5 4 3 7 6
code: $2, 7|3, 1, 4$
missing: 5, 6
1 2 5 4 3 7 6
code: $7|3, 1, 4, 5$
missing: $2, 6$
1 2 5 4 3 7 6
code: $|3, 1, 4, 5, 2$
missing: $6, 7$



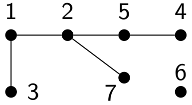
code: $5, 2, 7|3, 1$
missing: $4, 6$



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

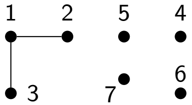


code: $7|3, 1, 4, 5$
missing: $2, 6$

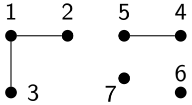


code: $|3, 1, 4, 5, 2$
missing: $6, 7$

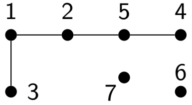
3. Recurse until you've cycled the bar to the front. Then draw and edge between the two numbers that are missing from your code.



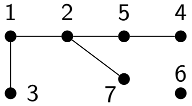
code: $5, 2, 7|3, 1$
missing: $4, 6$



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

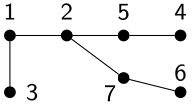


code: $7|3, 1, 4, 5$
missing: $2, 6$

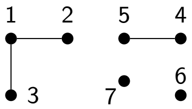


code: $|3, 1, 4, 5, 2$
missing: $6, 7$

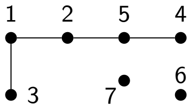
3. Recurse until you've cycled the bar to the front. Then draw and edge between the two numbers that are missing from your code.



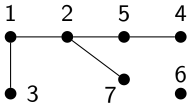
Done!
This is the tree!



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

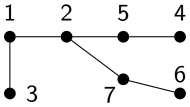


code: $7|3, 1, 4, 5$
missing: $2, 6$



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.

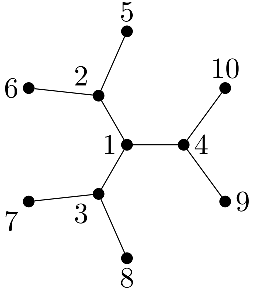


Done!
This is the tree!

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.

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 $\{1, \ldots, n\}$. □

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.