Overview

Puzzle

Attempt 1: Choosing edges

Attempt 2: Building up

Attempt 3

Prufer codes
Different format today. We’re going to look at one problem and we’ll see my failed attempts to solve it, along with a solution that actually works.
Counting trees

How many ways can we connect \( n \) vertices together into a tree?

Trees on 2 and trees on 3:
Trees on 4
What we know so far

Trees: Connected, acyclic.

<table>
<thead>
<tr>
<th>$n$</th>
<th>trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
</tr>
</tbody>
</table>
Choosing edges

Ok, first thought. A tree on $n$ vertices has $n - 1$ edges out of all possible edges:

$$\binom{n}{2} \binom{n}{n-1}.$$

<table>
<thead>
<tr>
<th>$n$</th>
<th>trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
</tr>
</tbody>
</table>

We counted cycles that aren’t trees.
Ways to add a vertex

So, let’s be careful to only generate trees. Here’s the thought. Consider a tree on $n$ vertices. We can add vertex $n + 1$ and connect it into the tree $n$ different ways. We’re guaranteed that the new graph is connected and acyclic (a tree!).

So, 2 vertices make 1 tree. Adding the 3rd vertex creates 2 times more. Adding the 4th vertex creates 3 times more.

Generalizing, we get $(n - 1)!$ trees.

<table>
<thead>
<tr>
<th>$n$</th>
<th>trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2 &lt; 3</td>
</tr>
<tr>
<td>4</td>
<td>6 &lt; 16</td>
</tr>
</tbody>
</table>

Now, we’re only counting trees, but we’re missing some trees. In particular, we’re missing trees where the last vertex is somewhere in the middle.
Generated 4 trees
More careful growing

When we add in vertex $n + 1$, the other $n$ vertices might not be a tree. They might be a forest. In general, vertex $n + 1$ will have one edge to each connected component in the forest. For the edge to connected component $i$, it will have a choice of which of the vertices in the connected component to connect to.

I worked on this path for awhile, and could almost write down an expression, but it was complicated and I didn’t think I could simplify it. The basic idea is consider all the ways of making a tree with $n'$ vertices for all $n' \leq n$, then all the ways $n$ vertices can be partitioned into clusters, then sum and multiply...

But even the question of how many partitions there are for $n$ items is hairy. See: https://oeis.org/A000110.
Better way to look at it

Sometimes there’s just a better way to look at it. It’s definitely not obvious (to me!). But it’s clever and gives a nice clean answer.

1. Create a “normal form” for trees. That way, we can at least notice when two different trees are actually the same.
2. Find a compact encoding for these trees. Here, “compact” means no extraneous information.
3. Then, we can show we have a bijection (!) between trees and the encoding.
4. If we’re lucky, it’ll be easier to count encodings than trees.
Normal form for trees

Choose vertex $n$ to be the root. Order children smallest (left) to biggest (right). There are no other choices.
Encoding a tree

Every vertex has a unique parent. So, we could just give the parent for each vertex: 3 7 4 5 11 3 10 2 10 4.

List $n - 1$ numbers (since root has no parent), each with a choice of $n - 1$ numbers (can’t pick yourself!). That’s $(n - 1)^{n-1}$.

Anything extraneous? Yes, can encode a loop: 3 1 2 5.
## Sanity check

Could it be \((n - 1)^{n-1}\)?

<table>
<thead>
<tr>
<th>(n)</th>
<th>trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>4 (&gt; 3)</td>
</tr>
<tr>
<td>4</td>
<td>27 (&gt; 16)</td>
</tr>
</tbody>
</table>

Yeah, we’re definitely overcounting. But we’re guaranteed not to undercount. Every tree has a representation in this scheme. But some representations do not produce trees. It’s not a bijection.
Prufer rule

Repeat the following procedure $n - 2$ times. Find the smallest valued leaf. Write down its parent. Delete the leaf.

3 3 4 2 7 10 10 4 5
Recover a tree

We can process any tree into a list. But can we recover the tree from the list? Before we prove that we can, let’s do an example:

3 3 4 2 7 10 10 4 5
4 by 4

Counting Trees

Prufer codes
Thinking inductively

Here’s a 6-vertex tree in Prufer encoding: 1 1 3 1.

In what sense is it built out of a 5-vertex encoding? Take the vertex $x$ that is the “first” leaf. Here, $x = 2$. Remove it, then renumber the vertices, decrementing anything larger than $x$. The thing to note is that the resulting tree and encoding still match!
Proof

Theorem: For every string $a \in [1, n]^{n-2}$ ($n \geq 2$), there is a unique tree $T$.

Proof: Build-down induction on $n$.

Base Case ($n = 2$): There is only one tree (a 1–2 segment) and only one encoding string (the null string).

Inductive Step ($n + 1$): Consider a string $a$ of length $n - 1$. In the tree $T$ encoded by $a$, the leaf with the smallest label $x$ must be linked to $a_1$.

Consider the string $a'$ formed by removing $a_1$ from $a$ and then subtracting one from every value in $a$ that’s larger than $x$. By the inductive hypothesis, there is a unique tree $T_0$ constructed from $a'$. We can construct a unique tree $T$ from $T_0$ by adding 1 to the values in $T_0$ that are $x$ or above, then adding the edge $(x, a_1)$. 
Summing up

We have a scheme for encoding trees as lists. It always works. We have a scheme for turning lists into trees. It always works. We have a bijection.

How many lists? \( n - 2 \) choices of numbers from 1 to \( n \). So, \( n^{n-2} \). Does that fit?

\[
\begin{array}{c|c}
 n & \text{trees} \\
\hline
2 & 1 \\
3 & 3 \\
4 & 16 \\
\end{array}
\]