Lecture 4: Combinatorial Arguments (Continued)

Date: February 2, 2026 **Scribe:** Jon Terry

The purpose of this lecture is to demonstrate that bijections can be quite useful in combinatorial proofs. But in order to conduct the demonstration, we need some definitions.

1 Trees

A *tree* $T = (V, E)$ is an undirected graph that is connected and acyclic. For example, the following graph is a tree.

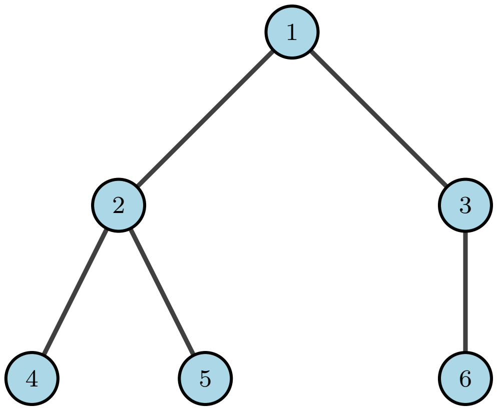

Figure 1: A tree.

The vertices 4, 5, and 6 are each called *leaves* of the tree. Formally, if $T = (V, E)$ is a tree, then $u \in V$ is a leaf of T if and only if $|\{v \in V : \{u, v\} \in E\}| = 1$.

1.1 Facts about Trees

Here are some facts about trees, presented without proof. For any tree $T = (V, E)$, the following statements are true:

- If $|V| \geq 2$, then T has at least two leaves.
- If $u \in V$ is a leaf of T, then $T' = T - u$ is a also a tree. In other words, removing a leaf from a tree yields a new tree (but with one less vertex than the tree we started with).
- $|V| = |E| + 1$. However the converse is not always true. In other words if $G = (V, E)$ is any graph with $|V| = |E| + 1$, G is not necessarily a tree.
- For all $u, v \in V$, there is a unique $\{u, v\}$-path in T. This fact follows almost immediately from the fact that T is connected and acyclic. Also, be aware that we often write $\exists!$ to stand for the phrase "there exists a unique . . . ."

2 Bijections

First, recall that given two sets X and Y, the function $f : X \to Y$ is a *bijection* between X and Y if f is *onto*, meaning that for all $y \in Y$ there is an $x \in X$ such that $f(x) = y$, and f is *one-to-one*, meaning that if $f(x_1) = f(x_2)$, then $x_1 = x_2$. Now, if f is a bijection, then we can define another function $f^{-1} = g : Y \to X$ given by $y = f(x) \mapsto x$. So another way to think about a bijection is to say that $f : X \to Y$ is a bijection if there exists a function $g : Y \to X$ such that $f \circ g = \mathrm{id}_Y$ and $g \circ f = \mathrm{id}_X$.

Now, here's what's neat about bijections and why they are useful in combinatorial proofs. Suppose we have a set A whose size (i.e. *cardinality*) we want to know, but this quantity is difficult to calculate. On the other hand, suppose we have a second set, B, whose cardinality is known (or at least easy to find). If we suspect that the cardinality of B and the cardinality of A are the same, then notice that if we can find a bijection between A and B, then $|A| = |B|$ and we're done!

3 Bijective Proofs

We can use this technique of constructing a bijection between sets to prove that there are n^{n-2} labeled trees on n vertices. Before we formally state the theorem, let's define our target set, B. Let B be the set of all strings of length $n-2$ we can form from an alphabet of $[n] \coloneqq \{1, 2, \ldots, n\}$ symbols. The claim is that there are n^{n-2} such strings. Why? Well, notice there are n choices for the first symbol of such a string, and n choices for the second symbol, and so on, up to the last $(n-2)$ symbol. Since these choices are independent of each other, then by the product principle discussed in Lecture 3, we have $|B| = n^{n-2}$.

Theorem 3.1. *For any $n \geq 1$, the number of labeled trees on n vertices is n^{n-2}.*

Proof. If $n = 1$, then the theorem is true, since $1^{1-2} = 1^{-1} = 1/1 = 1$ and a graph with one vertex and no edges is a tree trivially. So assume $n \geq 2$. Let $f : \mathcal{T} \to P([n], n-2)$, where $\mathcal{T}$ is the set of all trees on n vertices and $P([n], n-2)$ is the set of all strings of length $n-2$ that we can obtain from an alphabet of n symbols. We define f as Algorithm 1.

Algorithm 1: Prüfer Code Algorithm.

```
1 while $|V(T)| > 2$ do
2     Let $\ell'_i$ be the leaf of $T$ with the largest label
3     Let $w_i$ be the unique neighbor of $\ell'_i$
4     Remove $\ell'_i$ from $T$
5 return $(w_1, w_2, \ldots, w_{n-2})$
```

This called Prüfer's algorithm, and it gives us the map we want from the set of trees to the set of $n-2$-length strings. For example, consider the tree

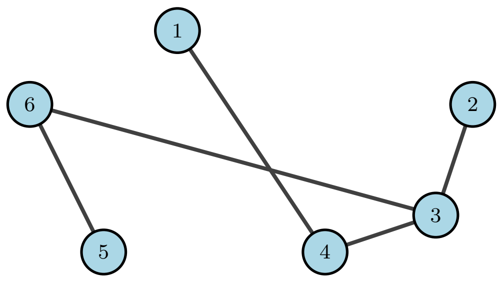

Figure 2: A tree on six nodes.

Applying Prüfer's algorithm to this tree produces the following output:

Table 1: Prüfer's algorithm on a six-node tree.

Step	Largest leaf (ℓ'_i) to delete	Neighbor (w_i)
1	5	6
2	6	3
3	2	3
4	3	4
5	Stop (only two vertices left).	–

Thus, the Prüfer sequence for the tree given above is 6 3 3 4.

Having established that $f : \mathcal{T} \to P([n], n-2)$, we define $g : P([n], n-2) \to \mathcal{T}$ as the inverse of the Prüfer algorithm. So, starting with a Prüfer sequence, g constructs a tree T' whose first $n-2$ edges are $\{\ell_i, w_i\}$ for all $i = \{1, \dots, n-2\}$, where

$$\ell_i = \max\{[n] \setminus \{\ell_1, \ell_2, \dots, \ell_{i-1}, w_i, w_{i+1}, \dots, w_{n-2}\}\},$$

and w_i is the ith vertex in the Prüfer sequence. The last edge of T' is found by connecting the two remaining vertices found in $\{[n] \setminus \{\ell_1, \dots, \ell_{n-2}\}\}$. In other words, g forms a tree from a Prüfer sequence by sequentially picking the largest *available* vertex (found by taking the list of all vertices and excluding from consideration all the vertices already used and those remaining in the sequence) and paring it with the current vertex in the Prüfer sequence. The final edge consists of the two vertices that were never deleted as leaves by f in the original process.

We can see how this works by applying the reverse algorithm to the Prüfer sequence from our example:

Table 2: Inverse Prüfer algorithm on the sequence 6 3 3 4

Step	Largest leaf (ℓ_i) to pick	w_i	Edge
1	$\ell_1 = \{[6] \setminus \{6,3,3,4\}\} = 5$	6	$\{5,6\}$
2	$\ell_2 = \{[6] \setminus \{5,3,3,4\}\} = 6$	3	$\{6,3\}$
3	$\ell_3 = \{[6] \setminus \{5,6,3,4\}\} = 2$	3	$\{2,3\}$
4	$\ell_4 = \{[6] \setminus \{5,6,2,4\}\} = 3$	4	$\{3,4\}$

Continuing with the proof, we claim that $g = f^{-1}$. Notice that f enumerated the edges of T to be

$$\{\{\ell'_1, w_1\}, \{\ell'_2, w_2\}, \dots, \{\ell'_{n-2}, w_{n-2}\}\} \cup \{[n] \setminus \{\ell'_1, \ell'_2, \dots, \ell'_{n-2}\}\},$$

while g enumerated the edges of T' as

$$\{\{\ell_1, w_1\}, \{\ell_2, w_2\}, \dots, \{\ell_{n-2}, w_{n-2}\}\} \cup \{[n] \setminus \{\ell_1, \ell_2, \dots, \ell_{n-2}\}\}.$$

Therefore, we need to show that $\ell'_i = \ell_i$ for all $i = \{1, 2, \dots, n-2\}$. To see why this is so, recognize that when we go to pick ℓ'_i we start with original set of $[n]$ vertices. But the vertices $\ell'_1, \ell'_2, \dots, \ell'_{i-1}$ have already been picked (so the algorithm cannot pick them again), and furthermore, we can't pick $w_i, w_{i+1}, \dots, w_{n-2}$ since they have degree ≥ 2 because they are neighbors of "future" leaves. Thus we have,

$$\ell'_i = \max\{[n] \setminus \{\ell'_1, \ell'_2, \dots, \ell_{i-1}, w_i, w_{i+1}, \dots, w_{n-2}\}\}.$$

Since this is exactly the set that defines ℓ_i for all $i = \{1, \dots, n-2\}$, the ℓ'_i's and ℓ_i's match. This establishes that f is bijection between the set of all trees on n vertices $\mathcal{T}$ and the set $P([n], n-2)$

of all strings of length $n-2$ that can be formed from an alphabet of n symbols. Therefore, since $|P([n], n-2)| = n^{n-2}$, then $|\mathcal{T}| = n^{n-2}$ which proves the claim. $\square$

The count in Theorem 3.1 is Cayley's formula (check out OEIS A000272).