Labeled Trees and Parking Functions

Directed Reading Program

Emma Yang

November 2024

1 A Preamble

I was never the biggest fan of trees—I remember finding them arbitrary and superfluous when first encountering them in Math 55, contrary to what I appreciated most about math. Yet, reading about combinatorics this semester and how trees fit into the whole picture has allowed me to see its connections with a plethora of other mathematical objects, and I have to admit that I can now appreciate them much more.

Many thanks to my mentor, Mistuki, for her patient, thorough, and adept guidance throughout this semester, both in DRP and Math 110, as well as my fellow mentee Jasmine. I enjoyed reading Sagan with you both!

2 Trees and Cayley's Formula

Trees are an important object in graph theory. They are also pretty interesting to count! In this section, I'll discuss labeled trees on the vertex set $[n] = \{1, 2, ..., n\}$ specifically, and one way we can count them. Before that, let's get some definitions out of the way.

Definition 2.1: A *tree* is a connected and acyclic graph.

Definition 2.2: A *labeled tree* is a tree in which each vertex is given a unique label.

The main theorem I will prove is as follows.

Theorem 2.1(Cayley's Formula): The number of labeled trees on n vertices is n^{n-2}.

There are many ways to prove this result, including the well-known Prüfer algorithm and Matrix

Tree Theorem. Here, I will prove the result using a generating function—algebraic objects associated with polynomials or series that make counting easier.

Proof: (Cayley's Tree Enumerator)[1]
Consider the set of labeled trees on the vertex set $[n] = \{1, 2, ..., n\}$.

Use $x_1, ..., x_n$ to represent each vertex, and $d_i(T)$ to denote the degree of each vertex x_i.

Definition 2.3: The *degree* of a vertex is the number of edges emanating from it.

Associate every tree T on the vertex set $[n]$ to the monomial

$$x_T = {x_1}^{d_1(T)} ... {x_n}^{d_n(T)}.$$

This is a neat way of associating trees with algebraic objects, especially because we can also interpret x_T as the product over all edges of T:

$$x_T = \prod_{\{i,j\} \in E(T)} x_i x_j.$$

This is because the variable occurs once in the product for every edge it is a part of; thus, the degree of each vertex-variable equals the number of edges it is associated with, which is precisely the degree of that vertex!

Then, the sum of the monomials, each representing a tree in the set of labeled trees, creates a generating function that can be written as follows:

$$\sum_T x_T = x_1 x_2 ... x_n (x_1 + x_2 + ... + x_n)^{n-2}.$$

Our goal is to show that this generating function in fact enumerates trees on $[n]$.

Theorem 2.1.1: The generating function enumerating trees on $[n]$ by the degrees of vertices is given by

$$\sum_T x_T = x_1 x_2 ... x_n (x_1 + x_2 + ... + x_n)^{n-2},$$

i.e.,

$$C_n(\mathbf{x}) = \sum_T x_T,$$

where $C_n(\mathbf{x})$ denotes the number of labeled trees on n vertices.

Proof:

We will prove that the polynomials $C_n(\mathbf{x})$ and $\sum_T x_T = x_1 x_2 ... x_n (x_1 + x_2 + ... + x_n)^{n-2}$ are the same by induction on n.

Base case: $n = 1$. There is one labeled tree on n, and $\sum_T x_T = x_1(x_1)^{1-2} = 1$. Thus $C_n(\mathbf{x}) = \sum_T x_T$ holds.

Inductive hypothesis: Suppose $C_n(\mathbf{x}) = \sum_T x_T$ holds for trees on $n-1$ vertices when $n > 1$.

Inductive step: Show $C_n(\mathbf{x}) = \sum_T x_T$ holds for trees on n vertices.

Notice that both polynomials contain some variable x_i to exactly the first power. This follows for $C_n(\mathbf{x})$ as every tree has a leaf—a vertex of degree 1; it follows for $x_1x_2...x_n(x_1 + x_2 + ... + x_n)^{n-2}$ because $(x_1 + x_2 + ... + x_n)^{n-2}$ has degree $n-2$, which implies that each of its terms omits at least one variable, while those variables still retain a first power from the preceding $x_1x_2...x_n$. Thus, we can fix an index i and consider only those terms containing x_i to the first power.

To extract these terms, we divide both polynomials by x_i, as every term contains x_i to at least the first power, then set $x_i = 0$ in the remaining terms. This way, we have kept the terms that originally contained x_i to the first power while terms with a higher power of x_i is turned into 0. We then want to show that for all i, the following holds:

$$(x_i^{-1}C_n(\mathbf{x}))_{x_i \mapsto 0} = (x_i^{-1}x_1 \dots x_n(x_1 + \dots + x_n)^{n-2}))_{x_i \mapsto 0}.$$

Without loss of generality, assume $i = n$. First, observe that for the RHS, we have

$$\begin{aligned} RHS &= (x_n^{-1}x_1 \dots x_n(x_1 + \dots + x_n)^{n-2}))_{x_n \mapsto 0} \\ &= x_1 \dots x_{n-1}(x_1 + \dots + x_{n-1})^{n-2} \\ &= (x_1 + \dots + x_{n-1})C_{n-1}(x_1, \dots, x_{n-1}), \end{aligned}$$

where the second equality follows from plugging in $x_n = 0$, and the third equality comes from the inductive hypothesis.

On the LHS, our procedure of dividing out by x_n and setting $x_n = 0$ leaves us with only the trees for which the vertex n was a leaf on the tree T, now each with the leaf n removed. To acquire the sum of these trees, we multiply both sides by x_n, thereby restoring the leaf n to the trees on the LHS, without recovering the trees for which n was not a leaf. Thus, we now want to show:

$$\sum_{T\text{: n is a leaf}} x_T = x_n(x_1 + \dots + x_{n-1})C_{n-1}(x_1, \dots, x_{n-1}),$$

i.e., that the RHS enumerates trees with vertex n as a leaf. To construct a tree for which n is a leaf, choose any tree T' on $[n-1]$, then connect any vertex in T' to n. The choice for T' is given by the generating function $C_{n-1}(x_1, \dots, x_{n-1})$, while the generating function for the choice of an edge i, n with $1 \leq i \leq n$ is $x_n(x_1 + \cdots + x_{n-1})$. Thus, $\sum x_{T^*}$, where T^* denotes a tree where n is a leaf, can be expressed as the product of these two generating functions, giving precisely the RHS. Because this equality holds for any vertex x_i which is a leaf, and equivalently, each monomial for which x_i has a power of 1, it means each of these monomials has the same coefficient on both sides. This implies $C_n(\mathbf{x}) = \sum_T x_T$. $\square$

Finally, to count the number of labeled trees on $[n]$, we simply plug in 1 for all the variables x_i as follows:

$$\begin{aligned} C_n(\mathbf{x}) &= x_1 x_2 ... x_n (x_1 + x_2 + ... + x_n)^{n-2} \\ &= 1 \cdot (n)^{n-2} \\ &= (n)^{n-2}, \end{aligned}$$

to get a total of n^{n-2} trees. $\square$

3 Cayley's Formula and Parking Functions

Something cool about Cayley's Formula is that the number n^{n-2} can also be found when counting other funky objects. One interesting example is the parking function.

Defintion 3.1: Parking Functions

Consider a procession of n cars, labeled from $A_1, ..., A_n$. They happen to be driving into a parking lot with n parking spaces, numbered $1, ..., n$, and each car A_i has a preferred parking spot p_i.

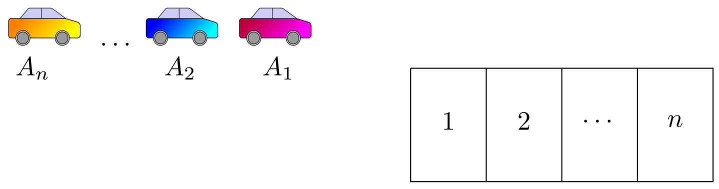


Fig. 3.1: The parking situation

As the cars drive through the parking lot, they park in their desired spot p_i if it's open; otherwise, they'll park in the next empty space if there is one, or leave the lot entirely. We call $p = (p_1, ..., p_n)$, the list of the cars' preferred spots, a *parking function of length* n if all cars manage to park.

Note that not all the cars have to park in their preferred spots in order for a list to be a parking function! In fact, the following claim illustrates the algebraic properties of a parking function:

Theorem 3.1: p is a parking function if and only if its unique weakly increasing rearrangement p'

satisfies $p'_i \leq i$ for all $n \in [n]$.

Proof [2]: First, let $p \in PF_n$ and let p' be its weakly increasing rearrangement. Assume there exists i such that $p'_i > i$. Then, there will be no car in parking spot i and consequently one car that will not be able to park. Therefore, $p'_i \leq i$ must hold.

To prove the converse, we pick p' as a weakly increasing vector that satisfies $p'_i = i$. This is clearly a parking function, as all cars will be parked in their desired spots. Moreover, decreasing the preferred spot for any car, i.e., making $p'_i < i$, will not result in it leaving the lot, as they can proceed to the next empty spot which exists because no spots are skipped as a result of $p'_i > i$. Any rearrangement of such a vector p' is still a parking function: if the preferences are distinct, all cars will park in their desired spots by the same logic; if not, rearranging p' will not change the fact that all cars *will* be able to park somewhere, and only affects the order in which the parking spots are taken and where each car ultimately parks. $\square$

A combinatorial question about these parking functions is how many of them there are for n hypothetical cars. We have the following result:

Theorem 3.2: The number of parking functions of length n is $(n+1)^{n-1}$.

Wow! The number of parking functions of length n happens to be exactly the number of labeled trees on $n+1$ vertices. This hints that we can try constructing a bijection between these two objects in order to prove Theorem 3.1. But before diving into that bijection, let's prove this result directly.

Proof 1 (The Division Rule): The idea here is to count a larger set of functions than what we are counting, then divide cleverly in order to get our desired number.

To achieve this, let's add another parking spot to our lot called "0", and rearrange the lot so that it is circular, like so:

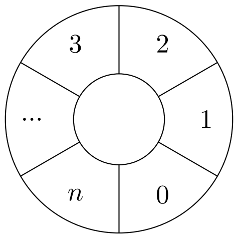


Fig. 3.2: New parking lot!

Now, instead of requiring cars to leave the lot once there are no more open spots, they are allowed to park in the 0^{th} spot, then the 1^{st} spot, and so on, following the circular arrangement of parking spots until every car is parked. Now, our n cars will have $n+1$ choices to park. This gives a total of $(n+1)^n$ possible sequences that denote the cars' desired parking spots: there are n slots for n

cars, and $n+1$ choices for each slot. Moreover, there will always be one spot open after all cars have parked, and a sequence will be a parking function if and only if the empty spot is the 0^{th} spot.

Notice that the sequences denoting the cars' desired spots are identical up to rotation, and each rotation simply changes which of the $n+1$ parking spots is empty—this means there are $n+1$ sets of identical sequences up to rotation. For instance, the sequence $(1,1,2)$ is the same as $(2,2,3)$, $(3,3,0)$, and $(0,0,1)$. It is easy to see that only one of the sequences in any group is an actual parking function—the one which corresponds to an empty 0^{th} spot (which would be $(1,1,2)$ in our previous example)! Thus, we are able to divide all cases, $(n+1)^n$, by $n+1$ to get the number of parking functions of length n, $(n+1)^{n-1}$. $\square$

The division rule is a technique we encountered early on in our counting career, and we were able to apply it in proving Theorem 3.2. However, this semester, I learned that constructing bijections is also an effective counting method: to count a set, simply biject it to a different set that you can already count, and they will automatically have the same cardinality.

Here, I will prove Theorem 3.2 by constructing a bijection between labeled trees on $n+1$ vertices, which we can count using what we proved in Section 2, and parking functions of length n.

Proof 2 (Labeled Trees Bijection):

Our goal is to construct a bijection between

(1) parking functions of length n, and

(2) labeled trees on $n+1$ vertices.

Denote the set of all parking functions of length n as $PF(n)$, and the set of all labeled trees on $n+1$ vertices as $\mathscr{T}_{n+1}$. Note that from Theorem 2.1, we have $|\mathscr{T}_{n+1}| = (n+1)^{n-1}$, by plugging in $n+1$ instead of n.

First, construct a function $f : \mathscr{T}_{n+1} \to PF(n)$, which takes a tree and outputs a parking function, defined by the following algorithm. (This initial construction of f is from Sagan[4] Chapter 1, Exercise 32, which we read this semester!)

Step 1: Let $T \in \mathscr{T}_{n+1}$, whose vertices are labeled $0,\dots,n$. Call vertex 0 the root of the tree. Draw T such that the root's *children* (vertices connected to the root) are in increasing order from left to right. Continue to do so for the children of the root's children (the *grandchildren*), etc.

Step 2: Create a permutation π by reading the children of the root from left to right, then the grandchildren of the root from left to right, and so on, until you reach the last vertex.

Step 3: Then, orient each edge of T so that it points from a vertex to its parent and call this set of arcs A.

Step 4: Map T to $p = (p_1, \dots, p_n)$ where

$$p_i = \begin{cases} 1 & \text{if } \vec{i0} \in A \\ 1+j & \text{if } \vec{i\pi_j} \in A \end{cases}$$

Hence, $f(T) = p \in PF(n)$ for some $T \in \mathscr{T}_{n+1}$.

Proof 2.1:

p is in fact a parking function because its weakly increasing rearrangement p' respects $p'_i \leq i$.

Any p'_i will either equal 1 or $1 + j$, where j is defined as in the algorithm above. There will always be at least one vertex k connected to 0 which results in $p_k = 1$, hence $p'_i \leq 1$. If $p'_i = 1 + j$, assume $1 + j > i$, which implies $j > i - 1$. j denotes the index of the vertex that the i^{th} vertex is a child of. However, this is impossible because j exceeds the number of already-existing vertices to which children can be added to, which is precisely $i - 1$, or the number of preceding numbers in the sequence (this will become more clear with the construction of the inverse of f). Thus, $p'_i \leq i$ must hold. $\square$

Example 4.1

We are able to get a parking function from the following labeled tree which we call T.

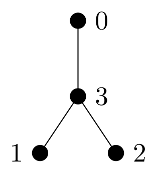


Fig. 3.3: A labeled tree T

We create a permutation $\pi = (312)$, by reading off the children and grandchildren of the root from left to right. Then, by orienting the edges such that each vertex points to its parent, we get a set of arcs $A = \{\widehat{30}, \widehat{23}, \widehat{13}\}$. We then map T to p, as follows:

p_1 should be 1+1 = 2, as the arc that starts from 1 is $\widehat{13}$, and 3 is the first number in our permutation, meaning $j = 1$.

p_2 should be $1 + 1 = 2$ as well, since $\widehat{23}$ is the corresponding arc, and $j = 1$ again.

p_3 should be 1, as the vertex 3 is the child of vertex 0 meaning that the arc is $\widehat{30}$.

Thus, T is mapped to $p = (2, 2, 1)$, which is indeed a parking function.

***

We then construct the inverse of f, called f', in order to show f is a bijection. f' should thus map elements of $PF(n)$ to elements in $\mathscr{T}_{n+1}$.

In the following construction, note that "vertex n" or the "n^{th} vertex" refers to the vertex whose index n is given by the number we get when counting (from 1) the children of vertex 0 from left to right, then the grandchildren of 0 from left to right, and so on (as we did in the algorithm for f). This index, which is crucial in the reconstruction and labeling of the tree, is distinct from the ultimate label the vertex will get (this process is detailed in Step 2).

Now, we can construct $f' : PF(n) \to \mathscr{T}_{n+1}$ as follows:

Step 1: Let $p \in PF(n)$. Rewrite p in its weakly increasing arrangement p'. We start with a 0 vertex.

For $i = 1, \ldots, n$, create a child of vertex $p'_i - 1$ for each p'_i. This gives the shape of the tree.

Step 2: Now, we refer to the original parking function p to label the remaining vertices of our tree.

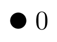
For $i = 1, \ldots, n$, label the child of the $(p_i - 1)^{th}$ vertex "i". Label each generation from left to right.

f' must output a tree, as our algorithm only adds children to existing vertices, which results in no cycles. Moreover, the tree will be on $n+1$ vertices, as we will create n new vertices from the n numbers in the parking function in addition to the original 0 vertex.

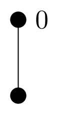
Example 4.2

Let's take the parking function $(2, 2, 1)$ that we got from applying f to the tree T in Example 4.1 and show that f' maps it back to T. Rewriting $(2, 2, 1)$ in its weakly increasing arrangement, we get $(1, 2, 2)$.

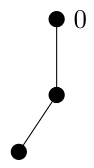
Start with the 0 vertex.



For $p'_1 = 1$, we create a child of the 0^{th} vertex, which we start with.



For $p'_2 = 2$, we create a child of the 1^{st} vertex—the vertex we've just created.



For $p'_3 = 2$, we create another child of that first vertex.

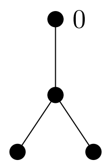
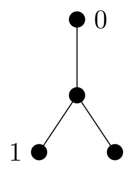


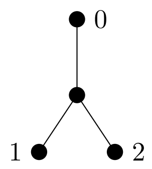
Fig. 3.4: Reconstructed tree

We have successfully reconstructed the shape of the tree! Now, let's label the vertices we've created using the original parking function $(2, 2, 1)$.

$p_1 = 2$, so we will label the child of the first vertex (counting the children of 0 from left to right, then the grandchildren and so on) as 1, starting from the leftmost child.



$p_2 = 2$, so we will label the next child of the first vertex as 2.



$p_3 = 1$, so we label the leftmost (and only) child of 0 as 3.

This is precisely T, the tree we started with!

***

To prove f' is the inverse of f, which will mean that f is bijective, we show that $f' \circ f$ and $f \circ f'$ are identity maps on their respective domains. This holds by construction, as f' was constructed by

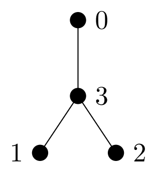


Fig. 3.5: Relabeled tree

reversing the steps of f, which itself is a well-defined function, as shown previously. Thus, both are well-defined functions mapping the same sets to each other in opposite directions.

Finally, $f : \mathscr{T}_{n+1} \to PF(n)$ is bijective, which implies $\mathscr{T}_{n+1}$ and $PF(n)$ have the same cardinality, i.e., $|PF(n)| = |\mathscr{T}_{n+1}| = (n+1)^{n-1}$. $\square$

Congrats, we've reached the end! To review, I applied a few of the counting techniques I've explored during my semester doing DRP in this write-up. Specifically, I counted labeled trees on n using a generating function, and parking functions of length n using the division rule and a bijection between the aforementioned labeled trees. I hope you learned something interesting, as I certainly have, and thanks for reading! :)

References

[1] Mark Haiman. *Notes on the Matrix-Tree theorem and Cayley's tree enumerator.* 2010.

[2] Jan Kretschmann. *Combinatorial problems related to optimal transport and parking functions.* 2023.

[3] Jeremy L. Martin. *What Else Can You Count If You Can Count Trees?* 2020.

[4] Bruce Sagan. *Combinatorics: The Art of Counting.* American Mathematical Society, Providence, 2020.

[5] Yufei Zhao. *Bijections.*