Parking functions and their applications in combinatorics

av

Selma Mohamed

2025 - No L11

Parking functions and their applications in combinatorics

Selma Mohamed

Självständigt arbete i matematik 15 högskolepoäng, grundnivå

Handledare: Per Alexandersson

2025

Abstract

Parking functions are intriguing combinatorial objects originally introduced to model linear probing in hashing and data storage. A parking function is a sequence of integers that encodes the preferences of n cars attempting to park in n spots along a one-way street. The sequence is valid if each car can successfully park by proceeding to its preferred spot or the next available one. This paper investigates parking functions from multiple perspectives, including their definition, enumeration, and bijective relationships with other mathematical structures such as rooted labeled trees, Dyck paths, and hyperplane arrangements. It also explores generalizations of parking functions and presents Python code for generating both sorted and unsorted variants.

Abstrakt

Parkeringsfunktioner är fascinerande kombinatoriska objekt som ursprungligen introducerades för att modellera linjär probing inom hashing och datalagring. En parkeringsfunktion är en heltalsföljd som representerar preferenserna hos n bilar som försöker parkera på n platser längs en enkelriktad gata. Följden är giltig om varje bil lyckas parkera genom att börja vid sin önskade plats eller fortsätta till nästa lediga. Denna uppsats undersöker parkeringsfunktioner ur flera perspektiv, inklusive deras definition, uppräkning och bijektiva samband med andra matematiska strukturer såsom rotade märkta träd, Dyck-stigar och hyperplansarrangemang. Vidare behandlas generaliseringar av parkeringsfunktioner och Python-kod presenteras för att generera både sorterade och osorterade varianter.

Contents

1 Introduction

What is a parking function? Imagine a one-way street with n parking spaces assigned numbers ranging from 1 to n, n cars arrive at the street one by one.

Definition 1. *A parking function refers to a sequence of preferences for* n *cars, each looking to park in one of* n *designated parking spots, numbered* $1, 2, \ldots, n$, *situated along a one-way street. Every car has a particular parking spot it prefers, and the parking process follows these rules:*

1. *Initially, each car attempts to park in its preferred spot.*
2. *If the preferred spot is already occupied, the car continues along the one-way street to the next available spot.*
3. *If a car cannot find any available spot before reaching the end of the street, the sequence is invalid.*

A sequence of preferences $(a_1, a_2, \ldots, a_n)$ *is called a parking function if all* n *cars can successfully park following these rules.*

Example 1.1. *An example of a parking function is (1,1,2). Car 1 preferred parking space is spot 1 and parks there, car 2 preferred parking space is also 1 so the car continues down the street to park in the first unoccupied spot which is spot 2, car 3 prefers spot 2 but it is occupied so the car parks in spot 3, which concludes all cars, so it is indeed a parking function. An example of a preference list that is not a parking function is (1,3,3). Car 1 prefers spot 1 and parks there, car 2 prefers spot 3 and parks there, lastly car 3 prefers spot 3 but it is occupied by car 2 so it cannot park, which means (1,3,3) is not a parking function.*

The mathematical significance of parking functions resides in their combinatorial characteristics and their links to various other structures [9].

Example 1.2. *For example, consider the sequence* $(3, 1, 7, 3, 1, 4, 1, 7)$ *representing the preferences of eight cars attempting to park in spots labeled* 1 *through* 8. *The parking process works as follows:*

Car 1 prefers spot 3 and parks there.

Car 2 prefers spot 1 and parks there.

Car 3 prefers spot 7 and parks there.

Car 4 prefers spot 3 which is occupied so it goes to spot 4 and parks there.

Car 5 prefers spot 1 (occupied), goes to 2 (free), and parks.

Car 6 prefers spot 4 (occupied), goes to 5 (free), and parks.

Car 7 prefers spot 1 (occupied), then goes to 2 (occupied), 3 (occupied), 4 (occupied), 5 (occupied), 6 (free), and parks.

Car 8 prefers spot 7 (occupied), then 8 (free), and parks.

Since all cars successfully find a spot, the sequence is a valid parking function.

1.1 Background

Parking functions were originally introduced by Konheim and Weiss [11] during their study of computer storage systems. They demonstrated, using analytical methods, that the number of parking functions of length n is given by the formula $(n+1)^{n-1}$. We denote by $\mathrm{PF}(n)$ the set of all parking functions of length n, so the number of parking functions is

$$|\mathrm{PF}(n)| = (n+1)^{n-1}.$$

This insight was later simplified by Pollak [8]. Parking functions are combinatorial structures that first arose during research on linear probing methods for resolving collisions in hash tables. Since then, they have been widely studied and found to be closely connected to other combinatorial objects, such as forests and hyperplane arrangements [7].

There are precisely 16 valid parking functions for 3 cars. These can be outlined as follows:

$$\begin{gathered}(1,1,1),\ (1,1,2),\ (1,1,3),\ (1,2,1),\ (1,2,2),\ (1,2,3),\\ (1,3,1),\ (1,3,2),\ (2,1,1),\ (2,1,2),\ (2,1,3),\ (2,2,1),\\ (2,3,1),\ (3,1,1),\ (3,1,2),\ (3,2,1).\end{gathered}$$

1.2 Hashing

A common and efficient technique used in computer science for storing and retrieving data is called *hashing* or the *scatter storage technique.* It uses a hash function h that assigns a specific hash value $h(K)$ to each item K. However, when two or more items receive the same hash value, a *collision* occurs. To handle such collisions, a method known as *linear probing and insertion* is used. This algorithm searches the hash table sequentially for the next available position. It assigns n items into $m > n$ empty cells labeled $0, 1, \ldots, m-1$, items are inserted one by one. For the k-th item, we attempt to place it in the first available cell in the sequence:

$$h_k, h_k + 1, h_k + 2, \ldots \pmod{m},$$

where h_k is the hash value of the k-th item and $0 \leq h_k < m$.

In this setup, the sequence $(h_1, \ldots, h_n)$ defines the *hash function.* It is said to be *confined* if, after inserting all items, the final cell $m-1$ remains unoccupied. Notably, when $m = n+1$, the confined hash functions correspond exactly to *parking functions* of length n. If an item is inserted into position p_k, then the total displacement is defined as:

$$D(\mathbf{h}) = \sum_{k=1}^{n} (p_k - h_k) \mod m,$$

which represents the total number of steps that the items have shifted from their original hash addresses. This displacement serves as a key statistic in the study of parking functions [9].

2 Proof of the Number of Parking Functions

Theorem 1. *The number of parking functions of length n, denoted by* $\mathrm{PF}(n)$*, is given by the formula*

$$f(n) = (n+1)^{n-1}.$$

The following proof is given in [2], though it is commonly attributed to Pollak (see [9]).

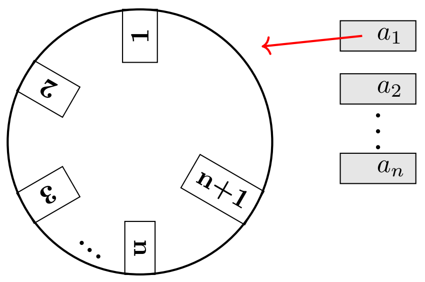
Proof 1. The proof for the number of parking functions uses a circular street analogy and exploits symmetry to reveal the relationship between *generic choice function* and parking functions. Now, imagine a circular one-way street with $n+1$ parking spots, and n cars entering the street. Each car has a preferred parking spot, chosen from the $n+1$ available spots. This setup creates a total of $(n+1)^n$ possible arrangements since each car can select from $n+1$ spots. The cars enter the street one at a time, starting with car 1. Each car parks in its preferred spot if it is available, and if the spot is occupied, the car continues driving clockwise until it finds an open spot. Because the street is circular, every car will eventually find a parking space, leaving exactly one spot empty at the end.

A *generic choice function* is considered a parking function if the empty spot happens to be parking spot $n+1$. The key idea is that, due to the symmetry of the circular street, the probability of any particular spot being the one left empty is equal for all $n+1$ spots. Since the empty spot could equally be any of the $n+1$ spaces, the number of parking functions is the total number of generic choice functions, $(n+1)^n$, divided by the number of possible empty spots, $n+1$. Thus, the number of parking functions is:

$$\frac{(n+1)^n}{n+1} = (n+1)^{n-1}.$$

□

Interestingly, this value, $(n+1)^{n-1}$, is also known to represent the number of rooted, labeled trees with $n+1$ vertices, where the root is labeled as vertex 0. In summary, the proof uses the circular street model to link all possible parking choices, referred to as *generic choice function*, with successful parking outcomes known as parking functions [2].



3 Proof of Parking Functions

Theorem 2. *A sequence $(a_1, a_2, \ldots, a_n)$ of positive integers is a parking function if and only if, when rearranged in non-decreasing order $(b_1 \leq b_2 \leq \cdots \leq b_n)$, it satisfies*

$$b_i \leq i \quad \textit{for all } i = 1, 2, \ldots, n.$$

Proof 2. We prove both directions of the equivalence.

Forward direction: Assume that $(a_1, \ldots, a_n)$ is a parking function. Then it must hold that $b_i \leq i$ for each i. For any index k, there must be at least k cars that prefer one of the parking spots in $\{1, 2, \ldots, k\}$. This directly leads to the inequality $b_k \leq k$ for all k.

Reverse direction: Now suppose that $b_i \leq i$ holds for every $i = 1, \ldots, n$. We aim to show that $(a_1, \ldots, a_n)$ is then a parking function. Assume instead that this is not the case. Then we may assume, by choosing a minimal counterexample, that the first $n-1$ cars successfully find parking, but the last car fails. This would imply that there exists a smallest index j such that all parking spots in $\{j, j+1, \ldots, n\}$ are taken after the first $n-1$ cars have parked, and the last car has $a_n \geq j$. Because the parking spot $j-1$ remains unoccupied, it follows that a total of $(n-j+1)+1$ cars attempted to park in spots $\geq j$. As a result, only $j-2$ cars were left wanting spots in $\{1, 2, \ldots, j-1\}$, which leads to the conclusion that $b_{j-1} > j-1$, contradicting our assumption [12]. □

3.1 Sorted parking functions

To enhance understanding of the concept, let's examine all the potential parking functions involving 4 cars. When it comes to 4 cars, there are precisely 14 sorted valid parking functions. These can be outlined as follows:

$$\begin{gathered}(1,1,1,1),\ (1,1,1,2),\ (1,1,1,3),\ (1,1,1,4),\\ (1,1,2,2),\ (1,1,2,3),\ (1,1,2,4),\ (1,2,2,2),\\ (1,2,2,3),\ (1,2,2,4),\ (1,2,3,3),\ (1,2,3,4),\\ (1,1,3,3),\ (1,1,3,4).\end{gathered}$$

Each of these meets the requirement that, the k-th element is no greater than k. This guarantees that all vehicles can park successfully, as established in Theorem 2.

Definition 2. *A* sorted parking function *is a parking function whose entries are in non-decreasing order. That is, a parking function $(a_1, a_2, \ldots, a_n)$ is sorted if it satisfies the characterization in Theorem 2, where $a_1 \leq a_2 \leq \cdots \leq a_n$.*

4 Bijection Between Parking Functions and Dyck Paths

Parking functions can also be constructed geometrically from Dyck paths. A *Dyck path* of order n is a lattice path from $(0,0)$ to (n,n), composed of unit steps East and North, that never passes below the diagonal $y = x$. A *labeled Dyck path* is a Dyck path where the n north steps are labeled with the numbers 1 through n (representing cars). We draw the label immediately to the right of each North step along the Dyck path. These labels are arranged so that each column strictly decreases from top to bottom.

Theorem 3. *The total number of Dyck paths of order n is given by the Catalan number*

$$C_n = \frac{1}{n+1}\binom{2n}{n}.$$

[3]

It is a classical exercise in combinatorics to show that the total number of unique Dyck paths of length $2n$ is given by the Catalan number formula [5].

A parking function can be represented as a preference function $f : \{1, \ldots, n\} \to \{1, \ldots, n\}$, where $f(i) = j$ means that car i prefers parking spot j. For f to define a valid parking function, it must satisfy:

$$|f^{-1}(\{1, \ldots, i\})| \geq i \quad \text{for all } 1 \leq i \leq n,$$

meaning that at least i cars prefer parking spots numbered from 1 to i as established in Theorem 2, which guaranteed since the path is Dyck path [2].

Given a labeled Dyck path, the corresponding parking function can be read by identifying the column in which each label appears. If label i appears in column j, this means that car i prefers parking spot j, so we define $f(i) = j$. If $f(i) = j$, meaning car i prefers spot j, then there is a North step labeled i in the jth column of the Dyck path diagram. Conversely, given a parking function f, we place label i in column $f(i)$, arranging the labels in each column in strictly decreasing order from top to bottom to satisfy the Dyck path conditions [9]

To recover the parking function from the labeled Dyck path in Figure 1, we scan the diagram from left to right, column by column. In each column, we look for the labels written to the right of the North steps. The label 1 appears in the first column, which means car 1 prefers parking spot 1. The label 2 appears in the fifth column, so car 2 prefers parking spot 5. The labels 3 and 4 both appear in the second column, which means cars 3 and 4 both prefer spot 2. The label 5 appears in the third column, so car 5 prefers spot 3. Putting this together, the parking function is:

$$f = (1, 5, 2, 2, 3).$$

This is how the Dyck path encodes the preferences of each car in the form of a parking function. To construct the labeled Dyck path from the parking function $f = (1, 5, 2, 2, 3)$, we proceed as follows: We place one North step for each car (from car 1 to car 5), and we label each North step with the number of the car it represents. Since car 1 prefers spot 1, we place a North step labeled 1 in column 1. Car 2 prefers spot 5, so its label is placed in column 5. Cars 3 and 4 both prefer spot 2, so we place two North steps in column 2, labeled 3 and 4. The larger label (4) goes above the smaller one (3), to maintain strictly decreasing order from top to bottom. Car 5 prefers spot 3, so we place label 5 in column 3. Once all North steps are placed in their respective columns, in decreasing order within each column, we connect them with East steps in between to form a Dyck path, a path that never goes below the diagonal. The resulting diagram is the labeled Dyck path shown in Figure 1.

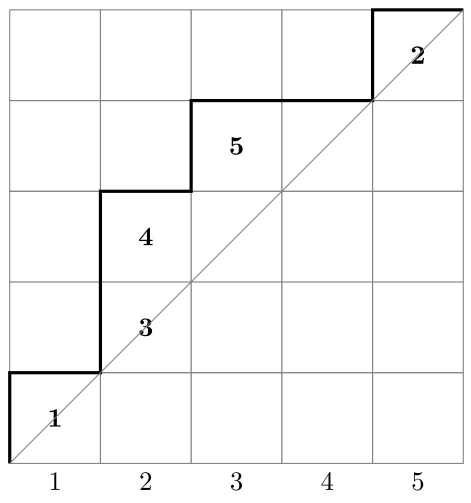


Figure 1: The labeled Dyck path of the parking function $f = (1, 5, 2, 2, 3)$.

4.1 Dyck Path and Rooted Labeled trees

As mentioned before Thereom 1 hints at a connection with rooted, labeled trees. We shall now explore this connection. The first step in this process is to associate each car with a node in a tree structure, where car i corresponds to a node labeled i. Next, we determine the children of each node, beginning with the root. The root node, labeled 0, is connected to the cars in the first column in the Dyck diagram. These cars serve as the children of the root node.

To identify the children of any other node labeled i, start at car i in the Dyck diagram and trace a path northeast at a 45-degree angle along the grid's diagonal. If this path intersects with another car at the bottom of a column, then all cars in that column become children of node i. If the path exits the grid without intersecting another car, then node i has no children (see Figures 3 and 4 for a concrete example) [2].

This bijective process leverages the geometric properties of parking functions represented through Dyck paths to establish a one-to-one correspondence with rooted labeled trees, thereby providing a concrete visual framework for understanding the structure of parking functions.

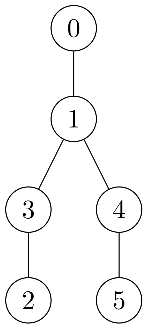


Figure 2: The rooted labeled tree corresponding to the parking function $f = (1, 5, 2, 2, 3)$ via the Dyck path in Figure 1.

Example 4.1. *To construct the rooted labeled tree from the Dyck path shown in Figure 3, we proceed as follows:*

Step 1. *Place cars along the Dyck path. Each number in the parking function sequence (e.g.,* $(3, 1, 7, 3, 1, 4, 1, 7)$*) represents a car's preferred parking spot. For each car i, place its label in the column corresponding to its preference* $f(i)$*, positioning the label to the right of a north step in that column. Multiple cars can appear in the same column if they share the same preference, and each column thus groups together the cars that prefer the same spot.*

Step 2. *Identify the root's children. All cars in the first column of the Dyck diagram (i.e., column 1) are connected to the root node labeled* 0*. These form the initial children of the root. In our example, cars* 2*,* 5*, and* 7 *appear in the first column, so:*

$$\textit{Children of } 0 : 2,\ 5,\ 7.$$

Step 3. *Trace diagonals to find further children. For each non-root node i, trace a* 45° *diagonal northeast from the position of car i in the Dyck diagram. If the path intersects another car at the bottom of a column, then every car in that column is assigned as a child of node i.*

> ***Node 2:*** *Its diagonal intersects boxes containing* 3 *and* 8*. These become children of* 2*.*
>
> $$\textit{Children of } 2 : 3,\ 8.$$
>
> ***Node 5:*** *Its diagonal intersects* 1 *and* 4*.*
>
> $$\textit{Children of } 5 : 1,\ 4.$$
>
> ***Node 4:*** *Its diagonal intersects* 6*.*
>
> $$\textit{Child of } 4 : 6.$$
>
> ***Node 7:*** *Its diagonal exits the grid without intersecting another car.*
>
> $$\textit{Children of } 7 : \textit{none}.$$

Step 4. *Assemble the tree. We now have the full tree:*

$$\begin{aligned}
0 &\to 2,\ 5,\ 7,\\
2 &\to 3,\ 8,\\
5 &\to 1,\ 4,\\
4 &\to 6.
\end{aligned}$$

See Figure 4.

[2]

4.2 Figures

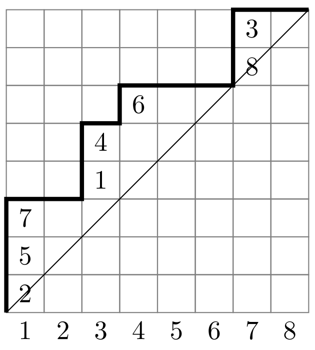


Figure 3: $(3, 1, 7, 3, 1, 4, 1, 7)$. Adapted from [2].

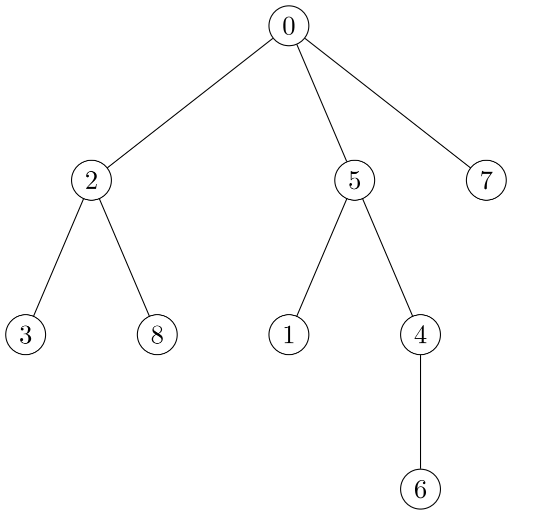


Figure 4: $(3, 1, 7, 3, 1, 4, 1, 7)$. Adapted from [2].

5 Bijection between Rooted Labeled Trees and Parking Functions

Theorem 4. *The number of rooted labeled trees on $n+1$ vertices is equal to the number of parking functions of length n, and is given by*

$$(n+1)^{n-1}.$$

This result follows from Cayley's formula, which counts rooted labeled trees, together with the enumeration formula for parking functions. Since both sets are counted by the same formula, it provides the existence of yet another natural combinatorial bijection between them, which will be constructed explicitly in the following proof.

Proof 3. We describe an explicit construction of a bijection.

Denote the set of rooted labeled trees with $n+1$ vertices as $\mathcal{T}_{n+1}$ and the set of parking functions of length n as $\mathrm{PF}(n)$. First, we define a function $f : \mathcal{T}_{n+1} \to \mathrm{PF}(n)$ by the following algorithm:

1. Let $T \in \mathcal{T}_{n+1}$, with vertices labeled $0, 1, \ldots, n$, where vertex 0 is considered the root. Draw the tree so that the children of each vertex are ordered increasingly from left to right.
2. Read the vertices level by level: first the children of the root from left to right, then the grandchildren from left to right, and so on, creating a permutation π of the non-root vertices $1, 2, \ldots, n$.
3. For each vertex i, orient each edge to point from the child to its parent, and define the set of *arcs* A.
4. Map T to the parking function $p = (p_1, \ldots, p_n)$ as follows:

$$p_i = \begin{cases} 1, & \text{if } i \to 0 \text{ is an arc in } A, \\ 1+j, & \text{if } i \to \pi_j \text{ is an arc in } A. \end{cases}$$

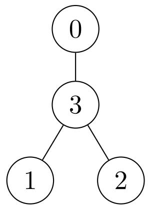
Hence, each tree $T \in \mathcal{T}_{n+1}$ uniquely corresponds to a parking function $p \in \mathrm{PF}(n)$. Since both sets $\mathcal{T}_{n+1}$ and $\mathrm{PF}(n)$ have the same cardinality, namely $(n+1)^{n-1}$, this construction defines a bijection.

□

Proof 3.1. We confirm that p is a parking function because its weakly increasing rearrangement p' satisfies $p'_i \leq i$. Each value p'_i is either equal to 1 or $1 + j$, where j is defined according to the construction method described earlier. There will always be at least one vertex k connected to 0, which gives $p_k = 1$, and so $p'_i \leq 1$. Suppose $p'_i = 1 + j$. Then $1 + j > i$, meaning $j > i - 1$. The value j represents the index of the vertex to which the i^{th} vertex is a child. However, this leads to a contradiction, because j would then be larger than the number of existing vertices that can have children, which is exactly $i - 1$, i.e., the number of earlier values in the sequence. Therefore, $p'_i \leq i$ must hold [10].

□

Example 5.1 (Recovering a parking function from a tree)**.** *We can derive a parking function from a labeled tree T using the function $\mathcal{T}_{n+1} \leftarrow f' : \mathrm{PF}(n)$.*



We begin by constructing the permutation π by reading the children and grandchildren of the root 0 from left to right. Thus, we have:

$$\pi = (3, 1, 2).$$

Next, we identify the directed arcs by pointing every child to its parent:

$$A = \{\overrightarrow{30}, \overrightarrow{23}, \overrightarrow{13}\}.$$

Following the rule to map T to a parking function $p = (p_1, p_2, p_3)$:

For p_1, the next number is 1. From arc $\overrightarrow{13}$, vertex 1 points to 3, which is the first element in π, so $j = 1$. Therefore, we apply $p_i = 1 + j$:

$$p_1 = 1 + 1 = 2.$$

For p_2, the last number is 2. From arc $\overrightarrow{23}$, vertex 2 points to 3, which is the first element in π, so again $j = 1$. Thus:

$$p_2 = 1 + 1 = 2.$$

For p_3, the first number in π is 3. Since 3 points to 0, according to the rule $p_i = 1$ if $i \to 0$, we have:

$$p_3 = 1.$$

Thus, the tree T is mapped to the parking function:

$$p = (2, 2, 1)$$

which satisfies the parking function conditions.

To construct the inverse $f' : \mathrm{PF}(n) \to \mathcal{T}_{n+1}$, follow this procedure:

1. *Given $p \in \mathrm{PF}(n)$, sort it into weakly increasing order p'. Start with a 0 vertex.*
2. *For $i = 1, \ldots, n$, create a child of vertex $p'_i - 1$. This gives the tree structure.*
3. *Use the original order of p to label the children. For each i, label the child of the $(p_i - 1)^{th}$ vertex "i" from left to right.*

Example 5.2 (Constructing a tree from a parking function)**.** *We apply the map $f' : \mathrm{PF}(n) \to \mathcal{T}_{n+1}$ to obtain a labeled tree on $n+1$ vertices from a given parking function.*

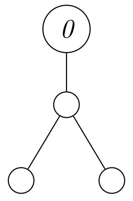
Let $p = (2, 2, 1)$. Then $p' = (1, 2, 2)$.

$p'_1 = 1$*: add a child to vertex 0.*

$p'_2 = 2$*: add a child to vertex 1.*

$p'_3 = 2$*: add another child to vertex 1.*

This gives the unlabeled rooted tree:



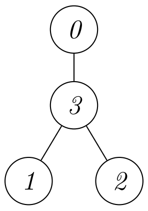
Now label using $p = (2, 2, 1)$:

$p_1 = 2$*: label first child of vertex 1 as 1.*

$p_2 = 2$*: label second child of vertex 1 as 2.*

$p_3 = 1$*: label child of 0 as 3.*

We get the relabeled rooted tree:



This matches the original tree. Thus, f' is the inverse of f, confirming the bijection [10].

We have established a bijective correspondence between rooted labeled trees with $n + 1$ vertices and parking functions of length n. This construction not only explains why both structures are enumerated by the same formula, but also provides a combinatorial interpretation that connects parking preferences directly to parent-child relationships within trees. The bijection confirms that each rooted labeled tree can be uniquely transformed into a parking function, and vice versa, thereby offering a deeper structural understanding of both combinatorial objects.

Example 5.3. *This tree corresponds to the same parking function as in Figure 4, but here it is obtained using the bijection $f' : \mathrm{PF}(n) \to \mathcal{T}_{n+1}$.*

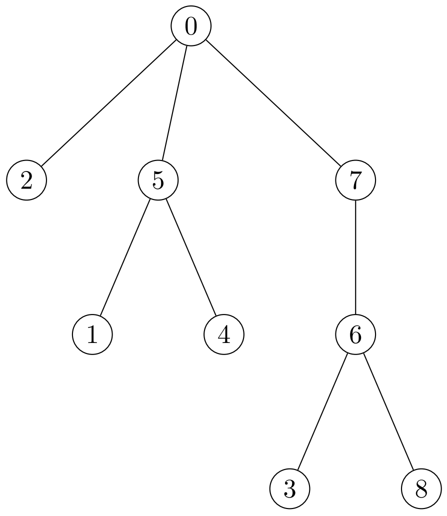


Figure 5: $(3, 1, 7, 3, 1, 4, 1, 7)$.

5.1 Another bijection using Breadth-first search (BFS)

In addition to the bijection described in Section 4.1 and Section 5, there exists another bijection between parking functions and rooted forests, based on the breadth-first search (BFS) algorithm. This bijection is given in [4], and provides a correspondence between rooted forests $F \in \mathcal{F}(m,n)$ and parking functions $f \in \mathrm{PF}(m,n)$.

From forest F to parking function $f \in \mathrm{PF}(m,n)$:

$$f_i = \begin{cases} j & p_i \text{ is a child of the root in the j:th tree (see Figure 6),} \\ (n-m+1) + \sigma_p(p_i) & \text{otherwise.} \end{cases}$$

where σ_p is the inverse BFS order on non-root vertices.

From parking function $f \in \mathrm{PF}(m,n)$ to forest F:

$$p_i = \begin{cases} 0j & f_i = j \text{ for some } j = 1, \ldots, n-m+1, \\ \tau_f^{-1}(f_i - (n-m+1)) & \text{otherwise.} \end{cases}$$

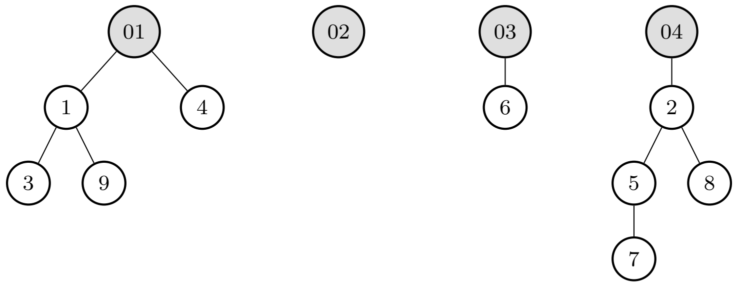


Figure 6: Forest F. Adapted from [4].

Example 5.4. *We apply the breadth-first search (BFS) to the forest F and read the vertices in BFS order, beginning with the root vertices and then proceeding level-wise to their children. At each level, vertices are ordered based on the label of their parent. Applying this method to F, we obtain:*

$$v_{01}, \ldots, v_{04}, v_5, \ldots, v_{13} = 01, 02, 03, 04, 1, 4, 6, 2, 3, 9, 5, 8, 7.$$

Removing the root vertices, we obtain the permutation:

$$\sigma_p^{-1} = (1, 4, 6, 2, 3, 9, 5, 8, 7),$$

and its inverse:

$$\sigma_p = (1, 4, 5, 2, 7, 3, 9, 8, 6).$$

We define the forest specification $t(p) = (r_1, \dots, r_{12})$, where each r_i denotes the degree of the vertex v_i (from v_{01} to v_{12}), excluding the final leaf v_{13}. This gives:

$$t(p) = (2, 0, 1, 1, 0, 2, 0, 2, 0, 0, 1, 0).$$

The specification $s(p)$ records how many times each parking spot is requested in the function $f \in \mathrm{PF}(9, 12)$, and it matches the forest specification:

$$s(p) = t(p) = (2, 0, 1, 1, 0, 2, 0, 2, 0, 0, 1, 0).$$

Using the non-decreasing rearrangement of $s(p)$, we obtain the sequence:

$$1^2, 3^1, 4^1, 6^2, 8^2, 11^1 = 1, 1, 3, 4, 6, 6, 8, 8, 11.$$

Before computing the parking function, we introduce the order permutation τ_f, *which records the position of each entry in the non-decreasing rearrangement of $s(p)$, preserving the left-to-right order among equal values. More precisely, $\tau_f(i)$ equals the position of $s(p)_i$ among the sorted values of $s(p)$.*

In our example, we have $\tau_f^{-1} = (1, 4, 6, 2, 3, 9, 5, 8, 7)$. We now replace each index i in $\tau_f^{-1}(i)$ with the ith smallest term in the sorted list:

Sorted list: 1, 1, 3, 4, 6, 6, 8, 8, 11.

$\tau_f^{-1}(1) = 1 \Rightarrow$ *First entry is 1, so* $f_1 = 1$

$\tau_f^{-1}(2) = 4 \Rightarrow$ *Fourth entry is 4, so* $f_2 = 4$

$\tau_f^{-1}(3) = 5 \Rightarrow$ *Fifth entry is 6, so* $f_3 = 6$

$\tau_f^{-1}(4) = 2 \Rightarrow$ *Second entry is 1, so* $f_4 = 1$

$\tau_f^{-1}(5) = 7 \Rightarrow$ *Seventh entry is 8, so* $f_5 = 8$

$\tau_f^{-1}(6) = 3 \Rightarrow$ *Third entry is 3, so* $f_6 = 3$

$\tau_f^{-1}(7) = 9 \Rightarrow$ *Ninth entry is 11, so* $f_7 = 11$

$\tau_f^{-1}(8) = 8 \Rightarrow$ *Eighth entry is 8, so* $f_8 = 8$

$\tau_f^{-1}(9) = 6 \Rightarrow$ *Sixth entry is 6, so* $f_9 = 6$

So we obtain:

$$f = (1, 4, 6, 1, 8, 3, 11, 8, 6).$$

[4]

6 Hyperplanes

A *hyperplane arrangement* is a finite collection of affine hyperplanes in $\mathbb{R}^n$. The *regions* of such an arrangement are the connected components that remain after all hyperplanes are removed from $\mathbb{R}^n$. A classical example is the *braid arrangement* A_n, which includes all hyperplanes of the form $x_i = x_j$ for $1 \leq i < j \leq n$. These hyperplanes correspond to the reflecting hyperplanes of the Coxeter group of type A_{n-1}, and their regions are in bijection with the permutations of the set $[n] = \{1, 2, \ldots, n\}$.

A *deformation* of A_n is a hyperplane arrangement in which every hyperplane is parallel to some hyperplane in the braid arrangement. One notable deformation is the *Shi arrangement*, denoted S_n, which consists of the hyperplanes

$$x_i - x_j = 0 \quad \text{and} \quad x_i - x_j = 1 \quad \text{for all } 1 \leq i < j \leq n,$$

dividing $\mathbb{R}^n$ into multiple regions. The Shi arrangement was introduced by Shi in the study of the affine Weyl group of type A_{n-1}, where group-theoretic methods were used to derive combinatorial results about these regions [6].

Theorem 5. *The number of regions of the Shi arrangement S_n is $(n+1)^{n-1}$.*

Parking functions have profound links with hyperplane arrangements. Each valid parking function coincides with a region in a hyperplane arrangement, contributing a geometric interpretation of the combinatorial counting problem [6].

Definition 3. *Let σ_n be the map from the regions of the Shi arrangement of type S_n to parking functions on $[n]$. This map sends a region with diagram ρ to a parking function f such that for each element i, the value $f(i)$ is equal to the position in ρ of the leftmost element in the chain that contains i.*

Theorem 6. *The map σ_n is a bijection between the regions of S_n and parking functions on $[n]$.*

Proof 4. We will explicitly describe the inverse of σ_n. Given a parking function f, we construct the partition π by grouping indices i and j into the same block whenever $f(i) = f(j)$. To form the chains, we arrange the elements within each block in increasing order from left to right. Next, we need to determine the permutation. We do this by placing the chains in sequence, based on the increasing order of their values under f. Let's assume we have already positioned the chains corresponding to values less than j, and now we focus on placing the chain with value j. Since f is a parking function, there are at least $j-1$ elements already positioned. We start by inserting the leftmost element of the chain into the j-th position, counting from the left. There is a unique way to arrange the remaining elements of the chain to the right, ensuring that no pair of arcs is formed where one arc contains another. This arrangement creates a braiding that gives rise to a diagram ρ, which corresponds to a specific region R within S_n [6]. $\square$

Example 6.1. *Start with the parking function:* $f = (6, 1, 6, 2, 2, 1, 2, 4, 1)$.

Step 1: *Group into chains (same value of $f(i)$)*

$f(i) = 1$*: {2, 6, 9}*

$f(i) = 2$*: {4, 5, 7}*

$f(i) = 4$*: {8}*

$f(i) = 6$*: {1, 3}*

Step 2: *Order the chains by increasing function value:*

$$\{2, 6, 9\}, \quad \{4, 5, 7\}, \quad \{8\}, \quad \{1, 3\}.$$

Step 3: *Place all elements from the chains left to right. The chains are positioned in increasing order of the function values $f(i)$. Within each chain, elements are listed in increasing numerical order. When arranging the elements, we ensure that no arc is nested within another; in other words, no arc lies entirely beneath another in the final arc diagram. This condition is crucial for building a valid braid diagram.*

A valid left-to-right placement of the elements is:

$$2, \ 4, \ 6, \ 8, \ 5, \ 1, \ 9, \ 7, \ 3.$$

Step 4: *Draw arcs between elements that belong to the same chain. These arcs reflect the hierarchical structure of the chains. This arc configuration defines the diagram of a Shi region. The inequality $x_i > x_j$ holds whenever element i lies to the left of j and they are connected by an arc.*

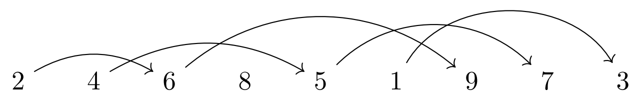


Figure 7: Arcs corresponding to chains in the permutation. Adapted from [6].

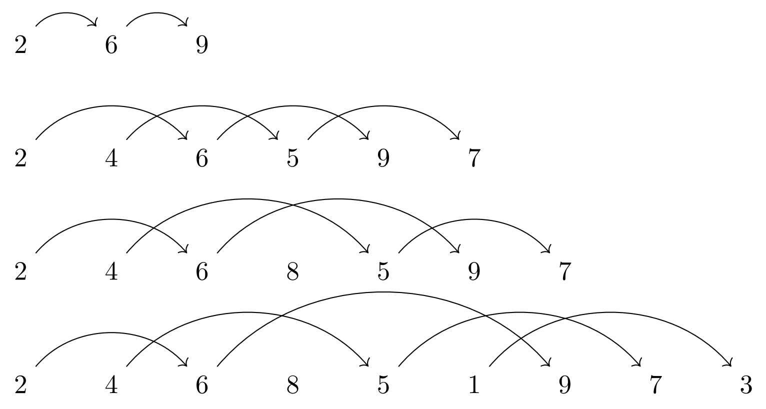


Figure 8: Constructing the region $\sigma_9^{-1}(f)$. Adapted from [6].

Example 6.2. *From Diagram to Parking Function*

Given permutation:

$$2,\ 4,\ 6,\ 8,\ 5,\ 1,\ 9,\ 7,\ 3$$

With arcs:

$$\{2,6,9\},\quad \{4,5,7\},\quad \{8\},\quad \{1,3\}$$

Find leftmost in each chain:

2 $\rightarrow$ *position 1* $\rightarrow f(2) = f(6) = f(9) = 1$

4 $\rightarrow$ *position 2* $\rightarrow f(4) = f(5) = f(7) = 2$

8 $\rightarrow$ *position 4* $\rightarrow f(8) = 4$

1 $\rightarrow$ *position 6* $\rightarrow f(1) = f(3) = 6$

Final reconstructed parking function: $f = (6, 1, 6, 2, 2, 1, 2, 4, 1)$

[6]

6.1 Figures

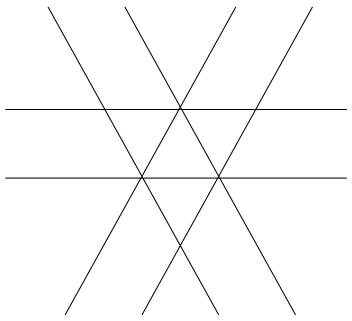


Figure 9: The Shi arrangement for $n = 3$. Reproduced from [6].

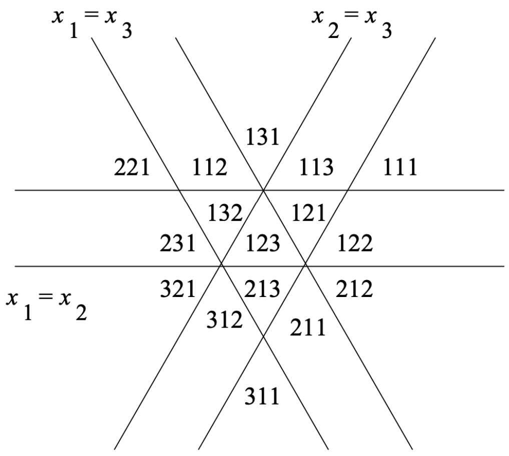


Figure 10: The bijection σ_3. Reproduced from [6].

7 Generalizations of Parking Functions

Parking functions, originating in combinatorics, transcend the traditional scenario of cars parking in sequentially numbered spaces. Their generalizations introduce innovative structures, including convoluted scenarios, leading to the development of mathematical structures with profound implications. In this section, we explore various generalizations of parking functions, highlighting their mathematical significance.

Movement variations in Parking Functions

Backward and forward movement of cars: Parking functions traditionally move forward along a one-way street until finding an available spot. However, one generalization allows cars to move backwards, this added flexibility fundamentally alters the nature of valid parking functions and introduces new combinatorial structures:

Single-step backward movement: If a car's desired parking spot is occupied, this generalization permits checking one spot behind before continuing down the street. This approach generates a distinct set of valid parking functions compared to the classical model and prompts questions about how limited backward movement affects the total number of valid functions.

Multiple-step backward movement: A more general version allows cars to move more than one spot backwards, following predetermined rules. For example, the number of steps a car can move backwards might depend on its position in the queue, or it could be permitted to jump a fixed number of spots backwards. This generalization results in a more nuanced enumeration problem, as it unlocks numerous possibilities for valid parking configurations.

Variable movement based on conditions: Another generalization allows certain cars to move backward while others can only move forward. This restriction can be based on factors such as a car's position in the parking sequence, for example, cars in odd-numbered positions may move backward while those in even-numbered positions cannot, or probabilistic rules, like a coin flip determining whether a car can move backward first. These variations create intricate patterns in parking behavior, making the enumeration and categorization of valid parking functions both demanding and mathematically stimulating [1].

Parking Functions with Cars of Different Sizes

In real-life scenarios, cars are not all the same size; some require more space than others. This observation inspires a generalization of parking functions, where cars can differ in length, reflecting their varying space requirements. Let us observe the following setup:

Variable-length cars: Presume each car c_i has a size $s_i \in \mathbb{N}$, requiring s_i contiguous parking spots to park. If its preferred spot is available but the adjacent spots needed are not, the car cannot park and must find an alternative arrangement. This version of the parking function is considerably more elaborate, as it introduces a spatial constraint that is absent in the traditional case.

Consecutive empty spots requirement: Discovering s_i consecutive open spots adds further complications to determining valid parking functions. This generalization is applicable to resource allocation problems that require contiguous resources to satisfy specific demands, making it a critical area of study in optimization and theoretical computer science.

Multiple cars occupancy of a single spot

In select variations, a single parking spot can accommodate more than one car. This generalization transforms the traditional parking functions into an expansive model, commonly referred to as "clown car functions".

Multiple cars per spot: One spot can contain up to a specified number of cars, say d. If a car's preferred spot is occupied, it moves forward to locate another available spot with sufficient space. This generalization is notably interesting because it leads to combinatorial problems that resemble queuing systems and resource-sharing models.

Variable capacities for spots: Building upon the concept further, one can examine spots with contrasting capacities, leading to even more complex variations of parking functions. Comparable generalizations find applications in distributed systems where resources differ in capacity and priority [1].

Obstructed and limited parking spots

Another interesting generalization emerges from introducing obstructions in the parking lot.

Obstructed spots: Particular spots can be blocked or temporarily unavailable due to external circumstances, such as roadblocks or maintenance work. The presence of obstructions incorporates a dimension of unpredictability, and valid parking functions must be reevaluated based on the available spots. This version has been studied to mimic real-life circumstances, such as heavy traffic and managing limited resources while following specific rules or restrictions.

Limited parking spots: There are instances where parking spots are fewer than cars, and in such situations, the goal shifts to maximizing the number of parked cars or finding ideal arrangements for parking. This scenario reflects practical problems in logistics and urban planning, where limited resources must be utilized effectively.

Randomness and probabilistic models in Parking Functions

Incorporating aspects of randomness to parking functions further cultivates their mathematical structure and complexity:

Random decisions: In this model, when a car encounters its preferred spot taken, it randomly decides whether to move backwards or forward (e.g., by flipping a coin). The probability that all cars can park successfully under these circumstances results in intriguing probabilistic analyses and combinatorial counting problems.

Probabilistic Parking Functions: Distinguishing the predicted behaviour of cars under random movement rules offers insights into the average-case intricacy of parking problems and has connections to probabilistic algorithms and Markov chains.

Teleporting Cars

An unusual but mathematically compelling variant involves teleporting cars. If a car's preferred spot is taken, it can be "teleported" to a different position, possibly based on a predetermined rule or randomly selected destination. This variant presents new combinatorial challenges and can be used to study allocation problems and unconventional queuing [1].

Structural Properties: Peaks, Valleys, and Other Patterns

Parking functions can also be generalized by highlighting structural properties within the list of preferences:

Peaks and Valleys: A peak occurs when a car's preferred spot is numerically greater than the spots preferred by its immediate neighbors in the preference list. Conversely, a valley occurs when a car's preferred spot is numerically smaller than the spots preferred by its immediate neighbors. For example, a peak is observed when a car prefers spot 4, while the cars before and after it prefer spots 2 and 3, respectively. The value 4 qualifies as a peak because it is greater than both its neighbors. Similarly, a valley occurs when a car prefers spot 2, while the cars before and after it prefer spots 4 and 3, respectively. The value 2 is considered a valley because it is smaller than both its neighbors. Counting and analyzing these features reveals deeper combinatorial structures and connects parking functions to permutation patterns and sorting algorithms.

Ascents, Descents, and Ties: Another approach to studying parking functions is counting the number of ascents, where a car prefers a higher spot than the previous one; descents, where it prefers a lower spot; and ties, where preferences are equal. These attributes offer insight into the ordering and arrangement properties of parking functions.

7.1 Conclusion

The generalization of parking functions outlined above showcases how an elementary combinatorial concept can evolve into a rich field of study with wide-ranging applications. From flexible movement rules and spatial constraints to unpredictability and structural arrangements, these variations enhance our understanding of combinatorial configurations and reveal new approaches for research. Exploring these generalizations not only amplifies theoretical combinatorics but also provides important insights for real-world applications in optimization, resource allocation, and algorithmic complexity [1].

8 Python codes

8.1 Code for parking functions

```python
from itertools import product

def is_parking_function(seq):
    """
    Check if a given sequence is a parking function.
    """
    seq = sorted(seq)
    for i, num in enumerate(seq, start=1):
        if num > i:
            return False
    return True

def generate_parking_functions(n):
    """
    Generate all parking functions of length n.
    """
    parking_functions = []
    # Generate all possible sequences of length n with
        values between 1 and n
    all_sequences = product(range(1, n + 1), repeat=n)

    for seq in all_sequences:
        if is_parking_function(seq):
            parking_functions.append(seq)

    return parking_functions

# Example usage
if __name__ == "__main__":
    n = int(input("Enter the length of parking functions (
        n): "))
    parking_functions = generate_parking_functions(n)
    print(f"All parking functions of length {n}:")
    for pf in parking_functions:
        print(pf)
```

8.2 Code for sorted parking functions

```python
from itertools import combinations_with_replacement

def is_sorted_parking_function(seq):
    """
    Check if a sorted sequence is a parking function.
    """
    for i, num in enumerate(seq, start=1):
        if num > i:
            return False
    return True

def generate_sorted_parking_functions(n):
    """
    Generate all sorted parking functions of length n.
    """
    parking_functions = []
    # Generate all non-decreasing sequences using
        combinations with replacement
    all_sequences = combinations_with_replacement(range(1,
         n + 1), n)

    for seq in all_sequences:
        if is_sorted_parking_function(seq):
            parking_functions.append(seq)

    return parking_functions

# Example usage
if __name__ == "__main__":
    n = int(input("Enter the length of parking functions (
        n): "))
    parking_functions = generate_sorted_parking_functions(
        n)
    print(f"All sorted parking functions of length {n}:")
    for pf in parking_functions:
        print(pf)
```

9 AI Statement

During the work on this project, I used AI tools for the following purposes:

Language assistance: I used ChatGPT to help polish English explanations.

Mathematical clarification: I consulted ChatGPT to confirm certain bijections and to explain complex concepts.

Python code support: I used ChatGPT to verify the correctness of the Python code and to improve code clarity and comments.

Typesetting: I used AI tools for help with LaTeX formatting and structuring.

References

[1] Carlson, J., Christensen, A., Harris, P. E., Jones, Z., & Ramos Rodríguez, A. (2020). *Parking Functions: Choose Your Own Adventure.*

[2] Haglund, J. (2004). *The q, t-Catalan Numbers and the Space of Diagonal Harmonics with an Appendix on the Combinatorics of Macdonald Polynomials.* Department of Mathematics, University of Pennsylvania.

[3] Liu, Y. (2024). *Exploring Parking Functions: Poset and Polytope Perspectives.* arXiv preprint.

[4] Yin, M. (2024). *Statistics of Parking Functions and Labeled Forests.* Department of Mathematics, University of Denver.

[5] Baracchini, G. (2016). *Dyck Paths and Up-Down Walks.* Undergraduate final paper, Massachusetts Institute of Technology.

[6] Athanasiadis, C. A., & Linusson, S. (1997). *A simple bijection for the regions of the Shi arrangement of hyperplanes.* arXiv preprint.

[7] Bruner, M.-L., & Panholzer, A. (2015). *Parking Functions for Trees and Mappings.* arXiv preprint.

[8] Li, Y., & Lin, Z. (2023). *A Symmetry on Parking Functions via Dyck Paths. Discrete Mathematics*, 346, 113426.

[9] Yan, C. H. (2015). *Parking Functions.* In M. Bóna (Ed.), *Handbook of Enumerative Combinatorics* (pp. 3–56). CRC Press.

[10] Yang, E. (2024). *Labeled Trees and Parking Functions.* Directed Reading Program, University of California, Berkeley, November 2024.

[11] Konheim, A. G., & Weiss, B. (1966). *An occupancy discipline and applications. SIAM Journal on Applied Mathematics*, 14(6), 1266–1274.

[12] Buczek, W. (2021). *Parking Functions.* Seminar talk, Theoretical Computer Science Department, Jagiellonian University.

Rättelse

1. “The following proof is due to Pollak, see [2]” **ändras till** “The following proof is given in [2], though it is commonly attributed to Pollak (see [9])”.

2. “If p_i = 0j for some…” **ändras till** “p_i is a child of the root in the j:th tree (see Figure 6)” **och** “where p_i is the parent of vertex i” **är borttagen.**

3. Källhänvisning av figurerna.

4. Referenserna är felaktigt numrerade, och en uppdaterad version kommer att skickas till Samuel senast på tisdag.