Parking Functions

- n cars, n parking spots on a one-way road.

Preference function \( f: [n] \rightarrow [n] \)

\[ f: i \mapsto f_i \quad f = (f_1, \ldots, f_n) \]

The driver of the \( i \)th car prefers to park in the \( f_i \)th parking spot.

- \( i \)th car drives to \( f_i \)th spot and parks there if the spot is empty.
  Otherwise, it keeps driving until it finds an empty spot and parks there.

Definition \( f = (f_1, \ldots, f_n) \) is called a parking function if all \( n \) cars will park.
Example \( n = 4 \)

\[ (x_1, x_2, x_3, x_4) = (3, 3, 1, 1) \]

\[ \Rightarrow \]

\[ \begin{array}{cccc}
1 & 2 & 3 & 4 \\
\end{array} \]

\[ \begin{array}{cccc}
1 & 2 & 3 & 4 \\
\end{array} \]

\[ \begin{array}{cccc}
1 & 2 & 3 & 4 \\
\end{array} \]

So \( (3, 3, 1, 1) \) is a parking function.

But \( (3, 3, 2, 2) \) is not a parking function, because the 4th car cannot park.
Lemma \( f = (f_1, \ldots, f_n) \quad f_i \in [n] \forall i \)

TFAE:

(A) \( f \) is a parking function

(B) The sequence \( f_1, \ldots, f_n \) contains

  at most 1 entry \( n \);

  at most 2 entries \( \geq n-1 \);

  at most 3 entries \( \geq n-2 \);

  \ldots

  at most \( k \) entries \( \geq n-k+1 \)

  for \( k = 1, 2, \ldots, n \).

\[ \# \{ f_i \geq n-k+1 \} \leq k \quad \text{for } k \in [n]. \]

(C) There exists a permutation

\( w_1, \ldots, w_n \) of \( 1, 2, \ldots, n \) s.t.

\( f_i \leq w_i \forall i = 1, 2, \ldots, n \)

Exercise. Prove that these 3 conditions are equivalent.

Note: It is clear that

\( (A) \implies (B) \)
Example $n = 3$.

All parking functions.

6 permutations:

1 2 3, 1 3 2, 2 1 3
2 3 1, 3 1 2, 3 2 1

and everything obtained from these permutations by decreasing some entries:

1 2 2, 2 1 2, 2 2 1
1 1 3, 1 3 1, 3 1 1
1 1 2, 1 2 1, 2 1 1
1 1 1

In total, $6 + 3 + 3 + 3 + 1 = 16$

parking functions for $n = 3$.

Theorem. There are exactly $(n+1)^n$ parking functions of size $n$. 
Proof. We will modify the setup as follows:

- $n$ cars
- $n+1$ parking spots on a circular road (with counter-clockwise direction)
- preference function $f : \{1, \ldots, n\} \rightarrow \{n+1\}$

The cars park according to the same procedure.

Now all $n$ cars will always park for any preference function $f : \{1, \ldots, n\} \rightarrow \{n+1\}$.

Moreover, one parking spot will never be left empty after all $n$ cars park.
This construction has circular symmetry:

If \( \tilde{f}_i = f_i + 1 \mod n+1 \)
for all \( i \in [n] \),

Then the resulting parking arrangement for \( \tilde{f} \) is obtained from the parking arrangement for \( f \) by shifting it 1 step counter-clockwise.

Example \( n = 3 \)

\[ f = (2, 3, 3) \]

\[ \tilde{f} = (2, 3, 3) \]

1st spot remains empty

\[ f = (3, 4, 4) \]

2nd spot remains empty

\[ f = (4, 1, 1) \]

3rd spot remains empty

\[ f = (1, 2, 2) \]

4th spot remains empty
Observation. A preference function \( f : [n] \rightarrow [n+1] \) is a parking function iff in the resulting parking arrangement of the cars the \((n+1)\)st spot remains empty.

This means that no car drives past the \((n+1)\)st parking spot. In other words, this is equivalent to parking on a non-circular road.

Let \( F_i \) be the set of preference functions for which \( i \)th spot remains empty.

\[
F_{n+1} = \text{the set of parking functions.}
\]

Because of the circular symmetry

\[
|F_1| = |F_2| = \ldots = |F_{n+1}|
\]

So

\[
\# \{ \text{parking functions} \} = |F_{n+1}|
\]

\[
\frac{1}{n+1} \sum_{f : [n] \rightarrow [n+1]} \#	ext{all preference functions}
\]

\[
= \frac{1}{n+1} \cdot (n+1)^n = (n+1)^{n-1}
\]

\( \square \)
Exercise Find a bijection between parking functions of size $n$ and spanning trees of $K_{n+1}$.

\[
\text{parking functions} \leftrightarrow \text{labelled trees}
\]

Dyck paths again

Consider a Dyck path $P$ of length $2n$ (i.e. $P$ has $n$ "up" steps and $n$ "down" steps)

Let us label all "up" steps in $P$ by $1, 2, \ldots, n$

(without repetitions) such that the labels in any consecutive sequence of "up" steps increase.

We'll call them labelled Dyck paths.

Example $n = 6$

\[\text{a labelled Dyck path}\]

Theorem The number of labelled Dyck paths of length $2n$ equals $(n+1)^{n-1}$.\\
Let's mark the Appeals so whose top steps can be linked by 1, 2, 5, at least once.

A labelled Dyck path $P$ corresponds to the partition function $f \left( \mathcal{E}, \mathcal{V}_d \right)$ at

$$d = 5$$

If the top steps labelled 1 is on the 5th diagonal.

Example (the above labelled), Dyck path $P$ is $f \left( \mathcal{E}, \mathcal{V}_d \right) = \left( \mathcal{E}, \mathcal{V}_d \right) = \left( \mathcal{E}, \mathcal{V}_d \right) = \left( \mathcal{E}, \mathcal{V}_d \right) = \left( \mathcal{E}, \mathcal{V}_d \right) = \left( \mathcal{E}, \mathcal{V}_d \right)$

It is also possible to consider the top steps labelled 1, 5

$$d = 5$$

It is also possible to consider the top steps labelled 1, 2

$$d = 5$$

We get

$$f \left( \mathcal{E}, \mathcal{V}_d \right) = \left( \mathcal{E}, \mathcal{V}_d \right) = \left( \mathcal{E}, \mathcal{V}_d \right) = \left( \mathcal{E}, \mathcal{V}_d \right) = \left( \mathcal{E}, \mathcal{V}_d \right) = \left( \mathcal{E}, \mathcal{V}_d \right)$$

Here vertex $i$, $v_i$ correspond to the labels in the Dyck path, and the values of $E_i$ correspond to diagonal in the Dyck path.

It is easy to see that the condition that $E_i$ is a perfect function correspond to condition that $P$ is a Pasha path.

Indeed, a lattice path $P$ then $(a_{i,j})$ is $(a_{i,j})$, which is a Dyck path if $P$ is

- no top step on OM line
- at least 1 top step on OM line
- at most 2 top steps on $(a_{i,j})$
- at least 1 top step on $(a_{i,j})$
- at most 2 top steps on $(a_{i,j})$

etc.

These conditions mean that

$$a_{i,j} = \begin{cases} \begin{array}{ll}
0 & \text{if } i < j \\
1 & \text{if } i = j \\
2 & \text{if } i > j
\end{array} \end{cases}$$

etc.

The last condition is (3) that defines partition functions.
Remark. Clearly, if we permute entries in a parking function, we get a parking function. For example

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \end{pmatrix} \mapsto \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \end{pmatrix} \mapsto \begin{pmatrix} 5 & 1 & 2 & 4 & 1 & 6 \end{pmatrix}$$

Permutations of $\binom{n}{m}$ correspond to relabellings of labelled Dyck paths.

Thus the equivalence classes of parking function under permutation of entries correspond to usual (unlabelled) Dyck paths.

Propositions 4 weakly increasing parking functions of size $n$ equals the $n^{th}$ Catalan number $C_n = \frac{1}{n+1} \binom{2n}{n}$.

Example $n = 3$.

$C_3 = 5$ weakly increasing parking functions:

$$123, 122, 113, 112, 111.$$
Generalized parking functions.

Fix \( n \geq 1 \) and \( \alpha = (\alpha_1, \ldots, \alpha_n) \) (where \( \alpha_i \)'s are positive integers)

**Definition**: \((f_1, \ldots, f_n)\) is an \( \alpha \)-parking function if:

- \( f_1, \ldots, f_n \in \{1, 2, \ldots, \alpha_n\} \)
- For all \( k = 1, 2, \ldots, n \)
  \[ \# \{ f_i \leq \alpha_k \} = k. \]

Equivalently, the set of \( \alpha \)-parking functions consists of all \((f_1, \ldots, f_n)\) obtained from \((\alpha_1, \ldots, \alpha_n)\) by

- permuting the entries,
  and/or
- decreasing some entries.

For example, the usual parking functions are exactly the \( \alpha \)-parking functions for \( \alpha = (1, 2, 3, \ldots, n) \).
Example \( n = 2 \) \( \alpha = (1, 4) \)

\( \alpha \)-parking functions:

14, 41, 13, 31, 12, 21, 11

7 \( \alpha \)-parking functions for \( \alpha = (1, 4) \).

**Theorem.** Fix \( n, k, \ell \geq 1 \).

Let \( \alpha = (\ell, \ell + k, \ell + 2k, \ell + 3k, \ldots, \ell + (n - 1)k) \)

Then \( \# \alpha \)-parking functions equals \( \ell \cdot (\ell + k \cdot n)^{n-1} \).

Example (above example)

\( n = 2, \ell = 1, k = 3 \)

\( \# \alpha \)-parking functions

\( = (1 + 2 \cdot 3)^{2-1} = 7 \).
Chip-firing game

(aka. Abelian sandpile model)

This is a simple mathematical model for complicated natural processes such as avalanches.

- [Per Bak, Chao Tang, Kurt Wiesenfeld, 1987]
- [Anders Björner, Laszlo Lovász, Peter Shor, 1991]

\[ G = (V, E) \] a connected finite graph with one selected vertex \( q \in V \), called the sink.

We'll assume \( V = \{0, 1, \ldots, m\} \) and \( q = 0 \).

A chip configuration \( C = (C_1, \ldots, C_m) \) is any non-negative integer vector.

We think of \( C_i \) as the number of chips at vertex \( i \).

Example \( G = \)

\[ \begin{array}{c}
\text{sink} \\
\cdots \\
\cdots \\
\end{array} \]

a chip configuration

\[ C = (4, 0, 2) \]

The arrow indicates chips at the sink \( q \).

The sink is a "black hole": any chip that goes into it "vanishes".
Let \( d_i = \deg G(i) \) degree of vertex \( i \).

If \( C_i \geq d_i \) then \( i \) is called a critical site. We can fire such vertex \( i \) by sending one chip to each neighbour of \( i \).

**Example.** We can fire vertex 3.

\[
C = (4, 0, 2) \rightarrow (5, 1, 0)
\]

Then we can fire vertex 1.

\[
(5, 1, 0) \rightarrow (2, 2, 1)
\]

Notice that one chip goes into the sink \( q \) and disappears. So the total number of chips decreases.

Now we cannot fire any vertex, because no chips at any vertex < the degree of the vertex.

**Def.** A chip configuration \((C_1, \ldots, C_n)\) is called stable if \( C_i < d_i \) \( \forall i \).

(i.e. we cannot fire any vertex)
Lemma. For any initial chip configuration $C_{\text{init}} = (G_1, \ldots, G_n)$,

- we will always obtain a stable chip configuration $C_{\text{stab}} = (G'_1, \ldots, G'_n)$ after a finite number of firings.
- The resulting stable configuration $C_{\text{stab}}$ is unique, i.e., it depends only on $C_{\text{init}}$ but not on a choice of firings.

$C_{\text{stab}}$ is called the **stabilization** of the initial config $C_{\text{init}}$.

Example (the previous example)

For $C_{\text{init}} = (4, 0, 2)$ the stabilization is $C_{\text{stab}} = (2, 2, 1)$.

We will arrive to the same stable configuration if we first fire vertex 1 and then fire vertex 3.
Consider the following random model

(1) Start with any initial configuration $C_{init}$

(2) Stabilize the configuration

(3) randomly select a vertex $i \in \{1, 2, \ldots, n\}$ (with uniform distribution) and add 1 chip to vertex $i$, i.e. increase $c_i$ by 1.

(3) go to step (1)

Basically, we keep repeating steps (2) and (1) (randomly drop a chip, stabilize, drop a chip, stabilize, etc.)
A stable configuration is called recurrent if it keeps occurring in the random process.

Theorem. A recurrent chip configuration equals the number of spanning trees of the graph $G$.

Example. $G = \begin{array}{c}
\text{4 vertices} \\
\text{2 edges}
\end{array}$

```
4 0 2 1 0 2 0 1
1 1

- drop a chip at vert. 2
- (Still a stable conf.)
- drop another chip at 1

2 1 0 2 0 1

- etc.
```

Recurrent configurations:
- $(1, 1)$, $(1, 0)$, $(0, 1)$

These are 3 spanning trees of $G$.

The configuration $(0, 0)$ is stable but not recurrent. It can appear only in initial steps of the random process. But if we run it for a while it can never occur.