Trees and Parking Functions 9-7-22

Last time:

Cayley's Tree Theorem: # of trees that connect n labeled vertices $1, 2, \ldots, n$ is n^{n-2}.

Pf 1: bijection between trees and Prüfer sequences (length $n-2$) by repeatedly removing smallest leaf and writing down its neighbor

Pf 2: bijection between trees and functions

$$f: \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$$

such that $f(1)=1$ and $f(n)=n$ by turning the tree into a directed graph and moving around edges to create some cycles.

Parking functions

Setup:

- n cars $1, 2, \ldots, n$
- n parking spaces $1, 2, \ldots, n$
- Each car has a preferred parking space
- The cars enter the parking lot one by one starting at space 1
  - If their preferred spot is available, they take it.
  - Otherwise, take the next available spot.
  - If no spots available after their preferred one, the car leaves without parking.
- The set of preferences is a parking function if all cars get to park.

Ex: Cars 1, 2, 3, 4, 5

$$\left.\begin{array}{l}\text{- car 1 prefers spot 3}\\ \text{- car 2 prefers spot 5}\\ \text{- car 3 prefers spot 3}\\ \text{- car 4 prefers spot 1}\\ \text{- car 5 prefers spot 4}\end{array}\right\} 3, 5, 3, 1, 4$$

What happens?

enter $\longrightarrow$ $\begin{array}{ccccc} 4 & & 1 & 3 & 2 \\ \hline 1 & 2 & 3 & 4 & 5 \end{array}$ $\longrightarrow$ exit

1. Car 1 takes spot 3.
2. Car 2 takes spot 5.
3. Car 3 drives to spot 3, sees it's taken, and takes spot 4.
4. Car 4 takes spot 1.
5. Car 5 drives to spot 4, sees it's taken. Keeps driving, but spot 5 is also taken, so car 5 leaves without parking.

This is not a parking function since car 5 didn't get to park.

Question: How many parking functions are there if we have n cars?

$n=1$: just $\boxed{1}$ parking function: 1 (car 1 prefers spot 1)

$n=2$:

$1\ 2 \longrightarrow \begin{array}{cc} 1 & 2 \\ \hline 1 & 2 \end{array}$ $\boxed{3}$ parking functions

$2\ 1 \longrightarrow \begin{array}{cc} 2 & 1 \\ \hline 1 & 2 \end{array}$

$1\ 1 \longrightarrow \begin{array}{cc} 1 & 2 \\ \hline 1 & 2 \end{array}$

~~2 2 → 1 2 can't park~~

~~1 2~~

<u>$n=3$:</u>

What are the parking functions for $n=3$, and how many?

<u>Claim:</u> A sequence is a valid parking function if and only if for any suffix, the # of cars with preferences in that suffix is $\leq$ the length of that suffix.

<u>Ex:</u> 3, 5, 3, 1, 4

<u>Cars preferring spots:</u>

<u>5:</u> car 2 (1 car) ✓

<u>4 or 5:</u> cars 2 and 5 (2 cars) ✓

<u>3, 4, or 5:</u> cars 1, 2, 3, 5 (<u>4</u> cars) ✗

<u>2, 3, 4, or 5:</u> cars 1, 2, 3, 5 (4 cars) ✓

<u>1, 2, 3, 4 or 5:</u> cars 1, 2, 3, 4, 5 (5 cars) ✓

<u>Problem:</u> 4 cars who all want the last 3 parking spots → not enough space for all 4 of them

Pf of claim: If more than k cars prefer the last k parking spots, then those k cars can't all park because all of them can only park in those last k spots (since they never turn back).

If there are never k cars who want the last k spots, then there will always be an unoccupied spot available for the next car.

Parking functions for $n=3$:

Invalid parking functions:

- all cars prefer spots 2 or 3 — 8 ways
- 2 cars prefer spot 3, one prefers spot 1 — 3 ways

All preference functions: 27 (3 choices for each car)

$27 - 11 = \boxed{16}$ parking functions

Conjecture: # of parking functions for n cars is $(n+1)^{n-1}$ = # of trees on $n+1$ vertices

Pf: Total # of preferences for n cars is n^n

Want to show that # parking function $(n+1)^{n-1}$

Idea: Say we have $n+1$ parking spaces on a circular track (still n cars).

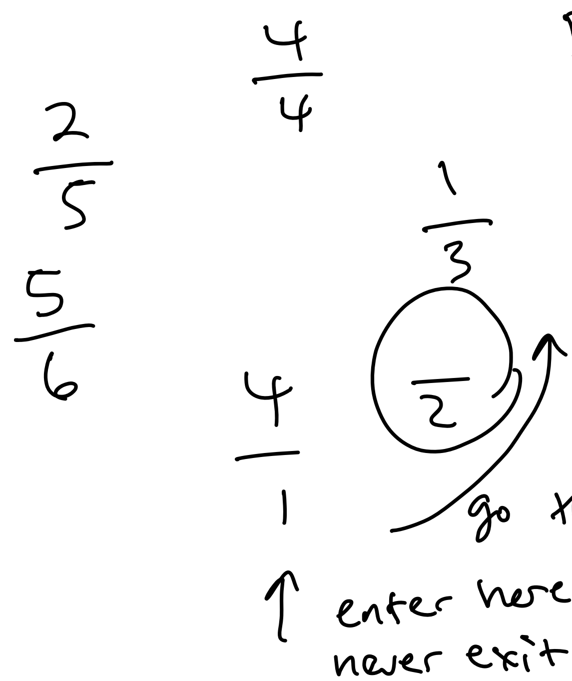
Now each car chooses a preferred spot $1, 2, \ldots, n+1$.

If their preferred spot is taken, keep driving in a circle so all cars always park, but there's a spot left open.

Now how many total preference functions are there?

$(n+1)^n$ total

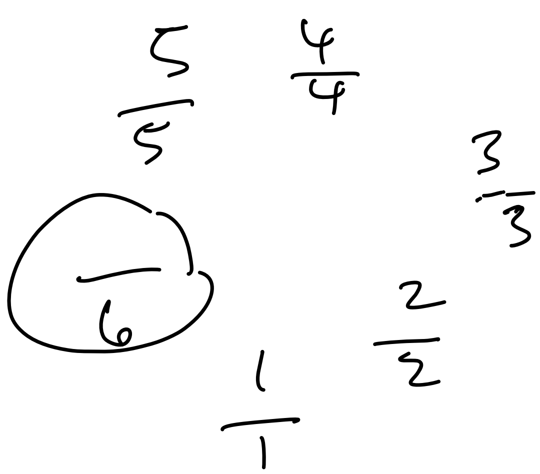
Which ones are parking functions in the original sense that the cars can park in spots $1, 2, \ldots, n$?



Preferences: 3, 5, 3, 1, 4
(not a parking function)

1. 1 takes spot 3
2. 2 takes spot 5
3. 3 takes spot 4
4. 4 takes spot 1
5. 5 takes spot 6

Spot 2 is left open!



Preferences: 1, 1, 1, 2, 2
(valid parking function)

1. 1 takes spot 1
2. 2 takes spot 2
3. 3 takes spot 3
4. 4 takes spot 4

5 takes spot 5

Spot 6 is left open.

We get a valid parking function in the original sense if and only if spot $n+1$ is the one left open.

Not a valid parking function $\Rightarrow$ no car leaves the first n spots, so $n+1$ left open.

Is a valid parking function $\Rightarrow$ some car does need to go past spot n, so the first such car takes spot $n+1$, and a different spot is left open.

Now what? Want to show that the # of preference functions where spot $n+1$ is the one left empty is $(n+1)^{n-1}$, knowing the total # of preference functions is $(n+1)^n$.

Claim: Each of the $n+1$ parking spots is equally likely to be left empty. (by symmetry?)

Idea: rotate a preference function $\to$ add a constant k, to all preferences, empty spot changes by k

To finish the proof: Since there are $n+1$ possible blank spots and they're all equally likely, the # of preference functions where $n+1$ is blank is

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

Question now: Why is the # of parking functions the same as the # of trees?

Ideas:

- Try to form a bijection between trees and circular paths:
  - Connect each node to its preferred parking space?

  Problem: might have loops!

  e.g. Car 1 prefers spot 1

  or Car 2 prefers spot 1 and car 1 prefers spot 2

Possible mapping:

Parking function ($n=11$):

2, 7, 1, 7, 2, 1, 8, 7, 8, 1, 5

Cars who prefer each spot:

1: 3, 6, 10

2: 1, 5

3: none

4: none

5: 11

6: none

7: 2, 4, 8

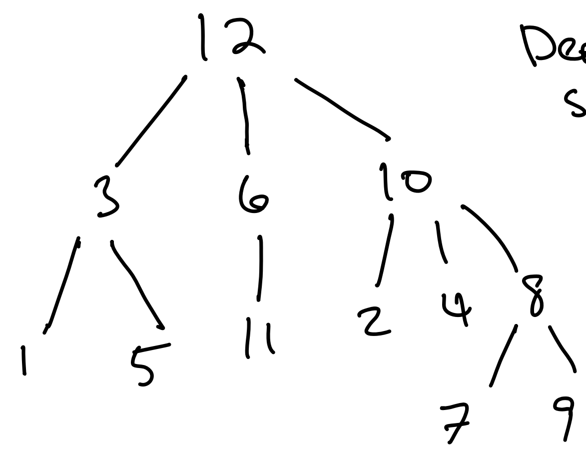
8: none

9: none

10: 7, 9

11: none

Build a tree:



Depth-first search order

order of nodes visited:

1. 12: 3, 6, 10
2. 3: 1, 5
3. 1: none
4. 5: none
5. 6: 11
6. 11: none
7. 10: 2, 4, 8
8. 2: none
9. 4: none
10. 8: 7, 9
11. 7: none

At ith step, visiting some node. Add as its children the cars who prefer spot i.

Parking function: # cars who prefer n is ≤ 1

cars who prefer $n-1$ or n is ≤ 2

cars who prefer $n-2, n-1$, or n is ≤ 3

$\vdots$

# of children of bottom right node is 0 (added last)

# of children of 2nd to last node visited is ≤ 1

# of children of 3rd to last node added is ≤ 2

(can only have as children the nodes not yet added)

At the kth to last step, there are $\leq k-1$ vertices not added yet, so the current node can have $\leq k-1$ children.