Hype, hype, hype, who did ~~win~~ become 2nd place?

Hype, hype, hype, who did ~~win~~ become 2nd place?

Let's discuss the problems first ;)

Team Building

- Given (n, k), compute n_1, n_2, n_3, n_4 such that

Team Building

- Given (n, k), compute n_1, n_2, n_3, n_4 such that
  1. $n_{i+1} = kn_i$

Team Building

- Given (n, k), compute n_1, n_2, n_3, n_4 such that
  1. $n_{i+1} = kn_i$
  2. $n_1 + n_2 + n_3 + n_4 = n$

Team Building

- Given (n, k), compute n_1, n_2, n_3, n_4 such that
  1. $n_{i+1} = kn_i$
  2. $n_1 + n_2 + n_3 + n_4 = n$
- We have

$$\begin{aligned} n &= n_1 + n_2 + n_3 + n_4 \\ &= n_1 + kn_1 + k^2 n_1 + k^3 n_1 \\ &= n_1(1 + k + k^2 + k^3) \end{aligned}$$

Team Building

- Given (n, k), compute n_1, n_2, n_3, n_4 such that
  1. $n_{i+1} = kn_i$
  2. $n_1 + n_2 + n_3 + n_4 = n$
- We have

$$\begin{aligned} n &= n_1 + n_2 + n_3 + n_4 \\ &= n_1 + kn_1 + k^2 n_1 + k^3 n_1 \\ &= n_1(1 + k + k^2 + k^3) \end{aligned}$$

- Hence $n_i = k^{i-1} \frac{n}{1+k+k^2+k^3}$

Incubator

We'll use multisets on this slide.

Incubator

We'll use multisets on this slide.

- Given a list of integers xs, compute

$$\max_{n \in \mathbb{N}} n \cdot \left|\{x \in xs \mid x \leq n\}\right|$$
$$= \max_{x \in xs} x \cdot \left|\{x \in xs \mid x \leq n\}\right|$$

Incubator

We'll use multisets on this slide.

- Given a list of integers xs, compute

$$\begin{aligned} &\max_{n \in \mathbb{N}} n \cdot |\{x \in xs \mid x \leq n\}| \\ =&\max_{x \in xs} x \cdot |\{x \in xs \mid x \leq n\}| \end{aligned}$$

- Naive solution: $\mathcal{O}(|xs|^2)$

Incubator

We'll use multisets on this slide.

- Given a list of integers xs, compute

$$\begin{aligned} &\max_{n \in \mathbb{N}} n \cdot |\{x \in xs \mid x \leq n\}| \\ =&\max_{x \in xs} x \cdot |\{x \in xs \mid x \leq n\}| \end{aligned}$$

- Naive solution: $\mathcal{O}(|xs|^2)$
- Better: first sort xs to lower to $\mathcal{O}\big(\log(|xs|)|xs|\big)$:

Incubator

We'll use multisets on this slide.

- Given a list of integers xs, compute

$$\begin{aligned} &\max_{n \in \mathbb{N}} n \cdot |\{x \in xs \mid x \leq n\}| \\ =&\max_{x \in xs} x \cdot |\{x \in xs \mid x \leq n\}| \end{aligned}$$

- Naive solution: $\mathcal{O}(|xs|^2)$
- Better: first sort xs to lower to $\mathcal{O}\big(\log(|xs|)|xs|\big)$:

```
startupRevenue l = aux (sort l) (length l)
  where
    aux [x] 1 = x
    aux (x:xs) n = max (x*n) (aux xs (n-1))
```

Tax Avoidance Scheme

Again, we'll use multisets.

[1] at least for all inputs we considered

Tax Avoidance Scheme

Again, we'll use multisets.

- Given $2 \leq a \leq 10^{10}$, compute $P \subseteq \mathbb{P}$ with $|P|$ minimal and $a = \sum_{p \in P} p$.

[1] at least for all inputs we considered

Tax Avoidance Scheme

Again, we'll use multisets.

- Given $2 \leq a \leq 10^{10}$, compute $P \subseteq \mathbb{P}$ with $|P|$ minimal and $a = \sum_{p \in P} p$.
- Goldbach tells us[1]: every even $n > 2$ is the sum of two primes. Hence:

[1] at least for all inputs we considered

Tax Avoidance Scheme

Again, we'll use multisets.

- Given $2 \leq a \leq 10^{10}$, compute $P \subseteq \mathbb{P}$ with $|P|$ minimal and $a = \sum_{p \in P} p$.
- Goldbach tells us[1]: every even $n > 2$ is the sum of two primes. Hence:
  1. If a is prime, return 1

[1] at least for all inputs we considered

Tax Avoidance Scheme

Again, we'll use multisets.

- Given $2 \leq a \leq 10^{10}$, compute $P \subseteq \mathbb{P}$ with $|P|$ minimal and $a = \sum_{p \in P} p$.
- Goldbach tells us[1]: every even $n > 2$ is the sum of two primes. Hence:
  1. If a is prime, return 1
  2. If a is even, return 2

[1] at least for all inputs we considered

Tax Avoidance Scheme

Again, we'll use multisets.

- Given $2 \leq a \leq 10^{10}$, compute $P \subseteq \mathbb{P}$ with $|P|$ minimal and $a = \sum_{p \in P} p$.
- Goldbach tells us[1]: every even $n > 2$ is the sum of two primes. Hence:
  1. If a is prime, return 1
  2. If a is even, return 2
  3. If a is odd...

[1] at least for all inputs we considered

Tax Avoidance Scheme

Again, we'll use multisets.

- Given $2 \leq a \leq 10^{10}$, compute $P \subseteq \mathbb{P}$ with $|P|$ minimal and $a = \sum_{p \in P} p$.
- Goldbach tells us[1]: every even $n > 2$ is the sum of two primes. Hence:
  1. If a is prime, return 1
  2. If a is even, return 2
  3. If a is odd... and a is the sum of two primes, one of the summands must be 2 since a is odd. Hence:
     3.1 If $a - 2$ is prime, return 2.

[1] at least for all inputs we considered

Tax Avoidance Scheme

Again, we'll use multisets.

- Given $2 \leq a \leq 10^{10}$, compute $P \subseteq \mathbb{P}$ with $|P|$ minimal and $a = \sum_{p \in P} p$.
- Goldbach tells us[1]: every even $n > 2$ is the sum of two primes. Hence:
  1. If a is prime, return 1
  2. If a is even, return 2
  3. If a is odd... and a is the sum of two primes, one of the summands must be 2 since a is odd. Hence:
     - 3.1 If $a - 2$ is prime, return 2.
     - 3.2 Otherwise return 3 since $a - 3$ is even.

[1] at least for all inputs we considered

Helping Rudolph

- Given a string $s \in \{L, R\}^n$, find the length of the longest substring that never moves outside the left boundary.

Helping Rudolph

- Given a string $s \in \{L, R\}^n$, find the length of the longest substring that never moves outside the left boundary.
- Idea: consider directions where you return to the original house without visiting any house left of the original house, e.g. *RRLRLL*. You can prepend or append this string to any valid list of directions. Compress those sequences.

Helping Rudolph

- Given a string $s \in \{L, R\}^n$, find the length of the longest substring that never moves outside the left boundary.
- Idea: consider directions where you return to the original house without visiting any house left of the original house, e.g. *RRLRLL*. You can prepend or append this string to any valid list of directions. Compress those sequences.
- Only valid sequences seperated by *R*s remain. Select and count the longest of those.

Helping Rudolph

```
compress s = foldl' compress1 [] s

compress1 s                          'R' = Open:s
compress1 (Comp k:Open:Comp n:s) 'L' = Comp (k + n + 2):s
compress1 (Open:Comp n:s)        'L' = Comp (n + 2):s
compress1 (Comp n:Open:s)        'L' = Comp (n + 2):s
compress1 (Open:s)               'L' = Comp 2:s
compress1 s                      'L' = Close:s
compress1 s                       _  = s
```

They See Me Slidin'

- Given list of stones, starting position, and a flag, can we reach the flag?

They See Me Slidin'

- Given list of stones, starting position, and a flag, can we reach the flag?
- Lists are too slow for lookups – use (Int)Maps and (Int)Sets to store stones for each row and column.

They See Me Slidin'

- Given list of stones, starting position, and a flag, can we reach the flag?
- Lists are too slow for lookups – use (Int)Maps and (Int)Sets to store stones for each row and column.
- Simply model walls as stones

They See Me Slidin'

- Given list of stones, starting position, and a flag, can we reach the flag?
- Lists are too slow for lookups – use (Int)Maps and (Int)Sets to store stones for each row and column.
- Simply model walls as stones
- Get next stone in each of the four directions using said maps.

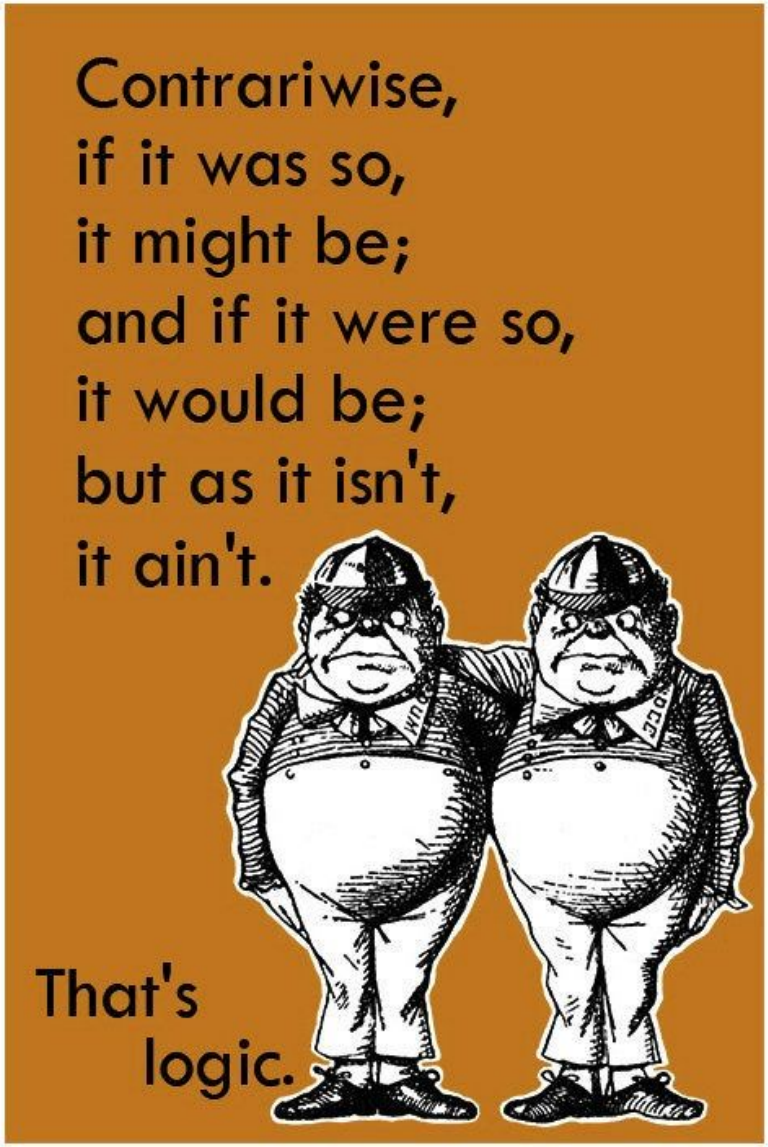
They See Me Slidin'

- Given list of stones, starting position, and a flag, can we reach the flag?
- Lists are too slow for lookups – use (Int)Maps and (Int)Sets to store stones for each row and column.
- Simply model walls as stones
- Get next stone in each of the four directions using said maps.
- Once we know the destination of the next slide, check if we will cross the flag.

They See Me Slidin'

- Given list of stones, starting position, and a flag, can we reach the flag?
- Lists are too slow for lookups – use (Int)Maps and (Int)Sets to store stones for each row and column.
- Simply model walls as stones
- Get next stone in each of the four directions using said maps.
- Once we know the destination of the next slide, check if we will cross the flag.
- Keep track of already visited positions in another Map to check for loops.

Terrible Poetry



Terrible Poetry

- Given a list of cards, decide if you need to flip the card to check the following rule:

Terrible Poetry

- Given a list of cards, decide if you need to flip the card to check the following rule:

  Every card with a golden heart has no even number on its opposite side.

Terrible Poetry

- Given a list of cards, decide if you need to flip the card to check the following rule:

  Every card with a golden heart has no even number on its opposite side.

- $A \to B \iff \neg A \vee B$; in our case:

Terrible Poetry

- Given a list of cards, decide if you need to flip the card to check the following rule:

  Every card with a golden heart has no even number on its opposite side.

- $A \to B \iff \neg A \vee B$; in our case:
  - $A \equiv$ golden heart

Terrible Poetry

- Given a list of cards, decide if you need to flip the card to check the following rule:

  Every card with a golden heart has no even number on its opposite side.

- $A \to B \iff \neg A \vee B$; in our case:
  - $A \equiv$ golden heart
  - $B \equiv$ odd number

Terrible Poetry

- Given a list of cards, decide if you need to flip the card to check the following rule:

  Every card with a golden heart has no even number on its opposite side.

- $A \to B \iff \neg A \vee B$; in our case:
  - $A \equiv$ golden heart
  - $B \equiv$ odd number
- If a heart is golden, we need to check the other side.

Terrible Poetry

- Given a list of cards, decide if you need to flip the card to check the following rule:

  Every card with a golden heart has no even number on its opposite side.

- $A \to B \iff \neg A \vee B$; in our case:
  - $A \equiv$ golden heart
  - $B \equiv$ odd number
- If a heart is golden, we need to check the other side.
- If a card is even, we need to check the other side.

Networking

- Find a greetings (edges) such that everybody is connected (transitively) at minimum cost. This is a minimum spanning tree problem.

Networking

- Find a greetings (edges) such that everybody is connected (transitively) at minimum cost. This is a minimum spanning tree problem.
- Use Kruskal's algorithm: sort edges by increasing cost. Start with an empty graph. For each edge add it to the graph if it does not lead to a cycle.

Networking

- Find a greetings (edges) such that everybody is connected (transitively) at minimum cost. This is a minimum spanning tree problem.
- Use Kruskal's algorithm: sort edges by increasing cost. Start with an empty graph. For each edge add it to the graph if it does not lead to a cycle.
- Check for cycles using a Union-Find data structure.

Networking

- Find a greetings (edges) such that everybody is connected (transitively) at minimum cost. This is a minimum spanning tree problem.
- Use Kruskal's algorithm: sort edges by increasing cost. Start with an empty graph. For each edge add it to the graph if it does not lead to a cycle.
- Check for cycles using a Union-Find data structure.
- Alternative: use Prim's algorithm.

Let's unfreeze the scoreboard!

Let's unfreeze the scoreboard!

Congratulations everyone!

Thanks for joining – see you on Friday!