

Problem Analysis Session

EUC 2025 judges

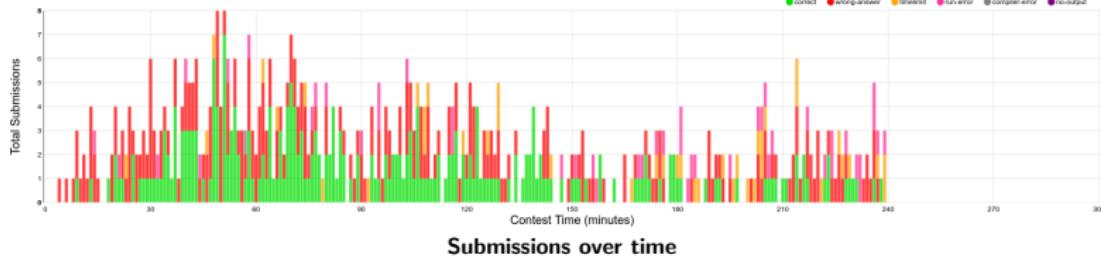
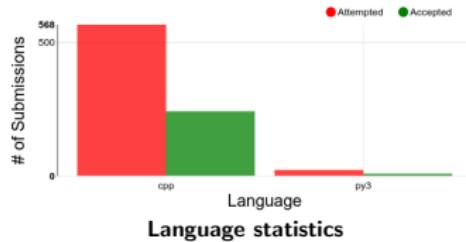
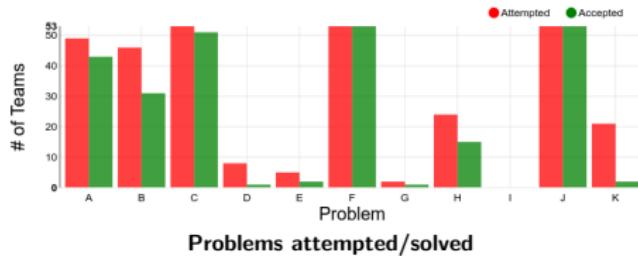
March 2, 2025

Our judges and problemsetters

- Federico Glaudo (Chief judge)
- Lucian Bicsi
- Andrea Ciprietti
- Jorke De Vlas
- Tomaz Hocevar
- Egor Kulikov
- Petr Mitrichev
- Pedro Paredes
- Giovanni Paolini
- Alice Sayutina

Statistics (after freeze)

Total number of submissions: 591
of which accepted: 252 ($\sim 42\%$)



Number of submissions: 76
of which accepted: 53



First solved by **Union University - Faculty of Computer Science**
after 20m



The problem

You are given n strings s_1, s_2, \dots, s_n , and a string t . Find (if it exists) a string that has s_1, \dots, s_n as subsequences but does not have t as a subsequence.

The problem

You are given n strings s_1, s_2, \dots, s_n , and a string t . Find (if it exists) a string that has s_1, \dots, s_n as subsequences but does not have t as a subsequence.

Solution

- If t is a subsequence of one s_i , then the answer is NO

The problem

You are given n strings s_1, s_2, \dots, s_n , and a string t . Find (if it exists) a string that has s_1, \dots, s_n as subsequences but does not have t as a subsequence.

Solution

- If t is a subsequence of one s_i , then the answer is NO
- Otherwise, construct the answer in a **greedy** fashion: remove the first character from one of the s_i and append it to the answer

The problem

You are given n strings s_1, s_2, \dots, s_n , and a string t . Find (if it exists) a string that has s_1, \dots, s_n as subsequences but does not have t as a subsequence.

Solution

- If t is a subsequence of one s_i , then the answer is NO
- Otherwise, construct the answer in a **greedy** fashion: remove the first character from one of the s_i and append it to the answer
- Give priority to characters different from the first character of t

The problem

You are given n strings s_1, s_2, \dots, s_n , and a string t . Find (if it exists) a string that has s_1, \dots, s_n as subsequences but does not have t as a subsequence.

Solution

- If t is a subsequence of one s_i , then the answer is NO
- Otherwise, construct the answer in a **greedy** fashion: remove the first character from one of the s_i and append it to the answer
- Give priority to characters different from the first character of t

Example

$s_1 = \text{abc}$, $s_2 = \text{cab}$, $s_3 = \text{aacbb}$

$t = \text{cbc}$

$ans =$

The problem

You are given n strings s_1, s_2, \dots, s_n , and a string t . Find (if it exists) a string that has s_1, \dots, s_n as subsequences but does not have t as a subsequence.

Solution

- If t is a subsequence of one s_i , then the answer is NO
- Otherwise, construct the answer in a **greedy** fashion: remove the first character from one of the s_i and append it to the answer
- Give priority to characters different from the first character of t

Example

$s_1 = \text{abc}$, $s_2 = \text{cab}$, $s_3 = \text{aacbb}$

$t = \text{cbc}$

$ans = \quad \leftarrow a$

The problem

You are given n strings s_1, s_2, \dots, s_n , and a string t . Find (if it exists) a string that has s_1, \dots, s_n as subsequences but does not have t as a subsequence.

Solution

- If t is a subsequence of one s_i , then the answer is NO
- Otherwise, construct the answer in a **greedy** fashion: remove the first character from one of the s_i and append it to the answer
- Give priority to characters different from the first character of t

Example

$s_1 = \text{abc}$, $s_2 = \text{cab}$, $s_3 = \text{aacbb}$

$t = \text{cbc}$

$\text{ans} = \text{a}$

The problem

You are given n strings s_1, s_2, \dots, s_n , and a string t . Find (if it exists) a string that has s_1, \dots, s_n as subsequences but does not have t as a subsequence.

Solution

- If t is a subsequence of one s_i , then the answer is NO
- Otherwise, construct the answer in a **greedy** fashion: remove the first character from one of the s_i and append it to the answer
- Give priority to characters different from the first character of t

Example

$s_1 = \text{abc}$, $s_2 = \text{cab}$, $s_3 = \text{aacbb}$

$t = \text{cbc}$

$ans = a \quad \leftarrow b$

The problem

You are given n strings s_1, s_2, \dots, s_n , and a string t . Find (if it exists) a string that has s_1, \dots, s_n as subsequences but does not have t as a subsequence.

Solution

- If t is a subsequence of one s_i , then the answer is NO
- Otherwise, construct the answer in a **greedy** fashion: remove the first character from one of the s_i and append it to the answer
- Give priority to characters different from the first character of t

Example

$s_1 = ab\textcolor{red}{c}$, $s_2 = \textcolor{red}{c}aba$, $s_3 = a\textcolor{red}{a}cbb$

$t = \textcolor{red}{c}bc$

$ans = ab$

The problem

You are given n strings s_1, s_2, \dots, s_n , and a string t . Find (if it exists) a string that has s_1, \dots, s_n as subsequences but does not have t as a subsequence.

Solution

- If t is a subsequence of one s_i , then the answer is NO
- Otherwise, construct the answer in a **greedy** fashion: remove the first character from one of the s_i and append it to the answer
- Give priority to characters different from the first character of t

Example

$s_1 = ab\textcolor{red}{c}$, $s_2 = \textcolor{red}{c}aba$, $s_3 = a\textcolor{red}{a}cbb$

$t = \textcolor{red}{c}bc$

$ans = ab \quad \leftarrow a$

The problem

You are given n strings s_1, s_2, \dots, s_n , and a string t . Find (if it exists) a string that has s_1, \dots, s_n as subsequences but does not have t as a subsequence.

Solution

- If t is a subsequence of one s_i , then the answer is NO
- Otherwise, construct the answer in a **greedy** fashion: remove the first character from one of the s_i and append it to the answer
- Give priority to characters different from the first character of t

Example

$s_1 = ab\textcolor{red}{c}$, $s_2 = \textcolor{red}{c}aba$, $s_3 = aa\textcolor{red}{c}bb$

$t = \textcolor{red}{c}bc$

$ans = aba$

The problem

You are given n strings s_1, s_2, \dots, s_n , and a string t . Find (if it exists) a string that has s_1, \dots, s_n as subsequences but does not have t as a subsequence.

Solution

- If t is a subsequence of one s_i , then the answer is NO
- Otherwise, construct the answer in a **greedy** fashion: remove the first character from one of the s_i and append it to the answer
- Give priority to characters different from the first character of t

Example

$s_1 = ab\textcolor{red}{c}$, $s_2 = \textcolor{red}{c}aba$, $s_3 = aa\textcolor{red}{c}bb$

$t = \textcolor{red}{c}bc$

$ans = aba \quad \leftarrow c$

The problem

You are given n strings s_1, s_2, \dots, s_n , and a string t . Find (if it exists) a string that has s_1, \dots, s_n as subsequences but does not have t as a subsequence.

Solution

- If t is a subsequence of one s_i , then the answer is NO
- Otherwise, construct the answer in a **greedy** fashion: remove the first character from one of the s_i and append it to the answer
- Give priority to characters different from the first character of t

Example

$s_1 = abc$, $s_2 = c\textcolor{red}{a}ba$, $s_3 = aac\textcolor{red}{b}b$

$t = \textcolor{gray}{c}\textcolor{red}{b}c$

$ans = abac$

The problem

You are given n strings s_1, s_2, \dots, s_n , and a string t . Find (if it exists) a string that has s_1, \dots, s_n as subsequences but does not have t as a subsequence.

Solution

- If t is a subsequence of one s_i , then the answer is NO
- Otherwise, construct the answer in a **greedy** fashion: remove the first character from one of the s_i and append it to the answer
- Give priority to characters different from the first character of t

Example

$s_1 = abc$, $s_2 = \text{c}\text{a}\text{b}\text{a}$, $s_3 = \text{a}\text{a}\text{c}\text{b}\text{b}$

$t = \text{c}\text{b}\text{c}$

$ans = abac \quad \leftarrow a$

The problem

You are given n strings s_1, s_2, \dots, s_n , and a string t . Find (if it exists) a string that has s_1, \dots, s_n as subsequences but does not have t as a subsequence.

Solution

- If t is a subsequence of one s_i , then the answer is NO
- Otherwise, construct the answer in a **greedy** fashion: remove the first character from one of the s_i and append it to the answer
- Give priority to characters different from the first character of t

Example

$s_1 = abc$, $s_2 = cab\textcolor{red}{a}$, $s_3 = aac\textcolor{red}{b}b$

$t = \textcolor{brown}{c}bc$

$ans = abaca$

The problem

You are given n strings s_1, s_2, \dots, s_n , and a string t . Find (if it exists) a string that has s_1, \dots, s_n as subsequences but does not have t as a subsequence.

Solution

- If t is a subsequence of one s_i , then the answer is NO
- Otherwise, construct the answer in a **greedy** fashion: remove the first character from one of the s_i and append it to the answer
- Give priority to characters different from the first character of t

Example

$s_1 = abc$, $s_2 = cab\textcolor{red}{a}$, $s_3 = aac\textcolor{red}{bb}$

$t = \textcolor{gray}{c}\textcolor{red}{b}c$

$ans = abaca \quad \leftarrow b$

The problem

You are given n strings s_1, s_2, \dots, s_n , and a string t . Find (if it exists) a string that has s_1, \dots, s_n as subsequences but does not have t as a subsequence.

Solution

- If t is a subsequence of one s_i , then the answer is NO
- Otherwise, construct the answer in a **greedy** fashion: remove the first character from one of the s_i and append it to the answer
- Give priority to characters different from the first character of t

Example

$s_1 = abc$, $s_2 = cab\textcolor{red}{a}$, $s_3 = aac\textcolor{red}{b}\textcolor{red}{b}$

$t = \textcolor{gray}{c}\textcolor{red}{b}\textcolor{red}{c}$

$ans = abacab$

The problem

You are given n strings s_1, s_2, \dots, s_n , and a string t . Find (if it exists) a string that has s_1, \dots, s_n as subsequences but does not have t as a subsequence.

Solution

- If t is a subsequence of one s_i , then the answer is NO
- Otherwise, construct the answer in a **greedy** fashion: remove the first character from one of the s_i and append it to the answer
- Give priority to characters different from the first character of t

Example

$s_1 = abc$, $s_2 = cab\textcolor{red}{a}$, $s_3 = aac\textcolor{red}{b}b$

$t = \textcolor{gray}{c}\textcolor{red}{b}\textcolor{red}{c}$

$ans = abacab \quad \leftarrow a$

The problem

You are given n strings s_1, s_2, \dots, s_n , and a string t . Find (if it exists) a string that has s_1, \dots, s_n as subsequences but does not have t as a subsequence.

Solution

- If t is a subsequence of one s_i , then the answer is NO
- Otherwise, construct the answer in a **greedy** fashion: remove the first character from one of the s_i and append it to the answer
- Give priority to characters different from the first character of t

Example

$s_1 = abc$, $s_2 = caba$, $s_3 = aacb\textcolor{red}{b}$

$t = \textcolor{gray}{c}\textcolor{red}{b}\textcolor{red}{c}$

$ans = abacaba$

The problem

You are given n strings s_1, s_2, \dots, s_n , and a string t . Find (if it exists) a string that has s_1, \dots, s_n as subsequences but does not have t as a subsequence.

Solution

- If t is a subsequence of one s_i , then the answer is NO
- Otherwise, construct the answer in a **greedy** fashion: remove the first character from one of the s_i and append it to the answer
- Give priority to characters different from the first character of t

Example

$s_1 = abc$, $s_2 = caba$, $s_3 = aacb\textcolor{red}{b}$

$t = \textcolor{gray}{c}\textcolor{red}{b}\textcolor{red}{c}$

$ans = abacaba \quad \leftarrow b$

The problem

You are given n strings s_1, s_2, \dots, s_n , and a string t . Find (if it exists) a string that has s_1, \dots, s_n as subsequences but does not have t as a subsequence.

Solution

- If t is a subsequence of one s_i , then the answer is NO
- Otherwise, construct the answer in a **greedy** fashion: remove the first character from one of the s_i and append it to the answer
- Give priority to characters different from the first character of t

Example

$s_1 = abc$, $s_2 = caba$, $s_3 = aacbb$

$t = cbc$

$ans = abacabab$

The problem

You are given n strings s_1, s_2, \dots, s_n , and a string t . Find (if it exists) a string that has s_1, \dots, s_n as subsequences but does not have t as a subsequence.

Solution

- If t is a subsequence of one s_i , then the answer is NO
- Otherwise, construct the answer in a **greedy** fashion: remove the first character from one of the s_i and append it to the answer
- Give priority to characters different from the first character of t
- **Achtung!** The most naive implementation of this algorithm has time complexity $n \cdot (|s_1| + \dots + |s_n| + |t|)$, which is too slow

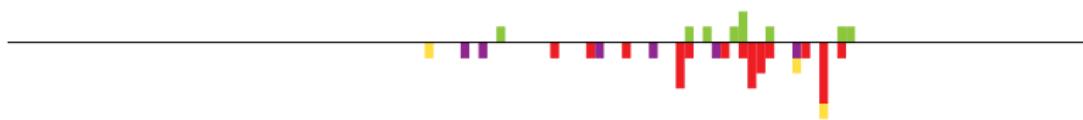
The Ultimate Wine Tasting Event

AUTHORED BY: Andrea Ciprietti PREPARED BY: Andrea Ciprietti

Number of **submissions**: 115
of which **accepted**: 53



First solved by **Masaryk University** after 13m



AUTHORED BY: Andrea Ciprietti PREPARED BY: Andrea Ciprietti

The problem

Given a sequence of $2n$ wines, n white and n red, can you split them into two subsequences of size n , swap them element-wise, and end up with all white wines on the left?

AUTHORED BY: Andrea Ciprietti PREPARED BY: Andrea Ciprietti

The problem

Given a sequence of $2n$ wines, n white and n red, can you split them into two subsequences of size n , swap them element-wise, and end up with all white wines on the left?

Solution

- Find an **equivalent condition** that is easy to verify

The problem

Given a sequence of $2n$ wines, n white and n red, can you split them into two subsequences of size n , swap them element-wise, and end up with all white wines on the left?

Solution

- Find an **equivalent condition** that is easy to verify
- Assuming it is possible, denote the two subsequences $a_1 < \dots < a_n$ and $b_1 < \dots < b_n$

The problem

Given a sequence of $2n$ wines, n white and n red, can you split them into two subsequences of size n , swap them element-wise, and end up with all white wines on the left?

Solution

- Find an **equivalent condition** that is easy to verify
- Assuming it is possible, denote the two subsequences $a_1 < \dots < a_n$ and $b_1 < \dots < b_n$
- Let's pick h and k such that:

The problem

Given a sequence of $2n$ wines, n white and n red, can you split them into two subsequences of size n , swap them element-wise, and end up with all white wines on the left?

Solution

- Find an **equivalent condition** that is easy to verify
- Assuming it is possible, denote the two subsequences $a_1 < \dots < a_n$ and $b_1 < \dots < b_n$
- Let's pick h and k such that:
 - a_h and b_k are in the first half;

AUTHORED BY: Andrea Ciprietti PREPARED BY: Andrea Ciprietti

The problem

Given a sequence of $2n$ wines, n white and n red, can you split them into two subsequences of size n , swap them element-wise, and end up with all white wines on the left?

Solution

- Find an **equivalent condition** that is easy to verify
- Assuming it is possible, denote the two subsequences $a_1 < \dots < a_n$ and $b_1 < \dots < b_n$
- Let's pick h and k such that:
 - a_h and b_k are in the first half;
 - a_{h+1} and b_{h+1} are in the second half (or either h or k is n);

AUTHORED BY: Andrea Ciprietti PREPARED BY: Andrea Ciprietti

The problem

Given a sequence of $2n$ wines, n white and n red, can you split them into two subsequences of size n , swap them element-wise, and end up with all white wines on the left?

Solution

- Find an **equivalent condition** that is easy to verify
- Assuming it is possible, denote the two subsequences $a_1 < \dots < a_n$ and $b_1 < \dots < b_n$
- Let's pick h and k such that:
 - a_h and b_k are in the first half;
 - a_{h+1} and b_{h+1} are in the second half (or either h or k is n);
 - $h \leq k$ (WLOG)

AUTHORED BY: Andrea Ciprietti PREPARED BY: Andrea Ciprietti

The problem

Given a sequence of $2n$ wines, n white and n red, can you split them into two subsequences of size n , swap them element-wise, and end up with all white wines on the left?

Solution

- Find an **equivalent condition** that is easy to verify
- Assuming it is possible, denote the two subsequences $a_1 < \dots < a_n$ and $b_1 < \dots < b_n$
- Let's pick h and k such that:
 - a_h and b_k are in the first half;
 - a_{h+1} and b_{h+1} are in the second half (or either h or k is n);
 - $h \leq k$ (WLOG)
- If $\{a\}$ has no bottles in the first half, set h to 0

Solution

- After swapping:

Solution

- After swapping:
 - bottles a_1, \dots, a_h and b_1, \dots, b_h end up in the first half, so they are white;

Solution

- After swapping:
 - bottles a_1, \dots, a_h and b_1, \dots, b_h end up in the first half, so they are white;
 - bottles b_{h+1}, \dots, b_k end up in the second half, so they are red

Solution

- After swapping:
 - bottles a_1, \dots, a_h and b_1, \dots, b_h end up in the first half, so they are white;
 - bottles b_{h+1}, \dots, b_k end up in the second half, so they are red
- \implies There are exactly $2h$ white wines in the first half

Solution

- After swapping:
 - bottles a_1, \dots, a_h and b_1, \dots, b_h end up in the first half, so they are white;
 - bottles b_{h+1}, \dots, b_k end up in the second half, so they are red
- \implies There are exactly $2h$ white wines in the first half
- Also, the first h wines are white

Solution

- After swapping:
 - bottles a_1, \dots, a_h and b_1, \dots, b_h end up in the first half, so they are white;
 - bottles b_{h+1}, \dots, b_k end up in the second half, so they are red
- \implies There are exactly $2h$ white wines in the first half
- Also, the first h wines are white
- By symmetry, the second half has $2h$ red wines and the last h are red

Solution

- After swapping:
 - bottles a_1, \dots, a_h and b_1, \dots, b_h end up in the first half, so they are white;
 - bottles b_{h+1}, \dots, b_k end up in the second half, so they are red
- \implies There are exactly $2h$ white wines in the first half
- Also, the first h wines are white
- By symmetry, the second half has $2h$ red wines and the last h are red
- These conditions are sufficient! **Exercise! :)**

C Ads

AUTHORED BY: Andrea Ciprietti PREPARED BY: Tomaz Hocevar

Number of submissions: 101
of which accepted: 51



First solved by **Ecole Polytechnique Fédérale de Lausanne** after
36m



C Ads

AUTHORED BY: Andrea Ciprietti PREPARED BY: Tomaz Hocevar

The problem

Minimize the number of shown ads by rearranging n videos of lengths d_i , where ads are shown between videos after every 3 videos or after $\geq k$ minutes of content.

C Ads

AUTHORED BY: Andrea Ciprietti PREPARED BY: Tomaz Hocevar

The problem

Minimize the number of shown ads by rearranging n videos of lengths d_i , where ads are shown between videos after every 3 videos or after $\geq k$ minutes of content.

Solution

- Ads partition the videos into groups → Minimize the number of groups

C Ads

AUTHORED BY: Andrea Ciprietti PREPARED BY: Tomaz Hocevar

The problem

Minimize the number of shown ads by rearranging n videos of lengths d_i , where ads are shown between videos after every 3 videos or after $\geq k$ minutes of content.

Solution

- Ads partition the videos into groups → Minimize the number of groups
- All *long* videos ($\text{length} \geq k$) go in different groups

C Ads

AUTHORED BY: Andrea Ciprietti PREPARED BY: Tomaz Hocevar

The problem

Minimize the number of shown ads by rearranging n videos of lengths d_i , where ads are shown between videos after every 3 videos or after $\geq k$ minutes of content.

Solution

- Ads partition the videos into groups → Minimize the number of groups
- All *long* videos ($\text{length} \geq k$) go in different groups
- Make as many groups of 3 videos as possible by finding pairs of *short* videos with total length $< k$

C Ads

AUTHORED BY: Andrea Ciprietti PREPARED BY: Tomaz Hocevar

The problem

Minimize the number of shown ads by rearranging n videos of lengths d_i , where ads are shown between videos after every 3 videos or after $\geq k$ minutes of content.

Solution

- Ads partition the videos into groups → Minimize the number of groups
- All *long* videos ($\text{length} \geq k$) go in different groups
- Make as many groups of 3 videos as possible by finding pairs of *short* videos with total length $< k$
- Pair up the remaining *short* videos with the remainng *long* videos, and any leftover *short* videos form groups of size 1

C Ads

AUTHORED BY: Andrea Ciprietti PREPARED BY: Tomaz Hocevar

The problem

Minimize the number of shown ads by rearranging n videos of lengths d_i , where ads are shown between videos after every 3 videos or after $\geq k$ minutes of content.

Solution

- Ads partition the videos into groups → Minimize the number of groups
- All *long* videos ($\text{length} \geq k$) go in different groups
- Make as many groups of 3 videos as possible by finding pairs of *short* videos with total length $< k$
- Pair up the remaining *short* videos with the remainng *long* videos, and any leftover *short* videos form groups of size 1
- **Complexity:** $O(n \log n)$ for sorting, then linear

A

Condorcet Elections

AUTHORED BY: Alice Sayutina PREPARED BY: Alice Sayutina

Number of submissions: 67
of which accepted: 43



First solved by **Ecole Polytechnique** after 9m



The problem

There is an election where each vote is a ranked list of candidates.
You are given n claims:

- “Candidate a_i is preferred to b_i by more than half voters”

Construct a valid list of votes.

The problem

There is an election where each vote is a ranked list of candidates.
You are given n claims:

- “Candidate a_i is preferred to b_i by more than half voters”

Construct a valid list of votes.

Solution

- The answer is always YES

The problem

There is an election where each vote is a ranked list of candidates.
You are given n claims:

- “Candidate a_i is preferred to b_i by more than half voters”

Construct a valid list of votes.

Solution

- The answer is always YES
- For a list of voters, let $\delta_{a,b}$ (voters) be how many votes prefer a to b minus how many votes prefer b to a

The problem

There is an election where each vote is a ranked list of candidates.
You are given n claims:

- “Candidate a_i is preferred to b_i by more than half voters”

Construct a valid list of votes.

Solution

- The answer is always YES
- For a list of voters, let $\delta_{a,b}$ (voters) be how many votes prefer a to b minus how many votes prefer b to a
- We need to find ans: $\delta_{a,b}(\text{ans}) > 0$ for all required (a, b)

Reminder

$\delta_{a,b}(\text{voters})$ is how many voters prefer a to b minus how many voters prefer b to a

Solution

- For every a, b , we construct votes π_1, π_2 such that:

Reminder

$\delta_{a,b}(\text{voters})$ is how many votes prefer a to b minus how many votes prefer b to a

Solution

- For every a, b , we construct votes π_1, π_2 such that:
 - $\delta_{a,b}(\{\pi_1, \pi_2\}) > 0$;

Reminder

$\delta_{a,b}(\text{voters})$ is how many votes prefer a to b minus how many votes prefer b to a

Solution

- For every a, b , we construct votes π_1, π_2 such that:
 - $\delta_{a,b}(\{\pi_1, \pi_2\}) > 0$;
 - δ for other pairs of candidates is zero

Reminder

$\delta_{a,b}(\text{voters})$ is how many votes prefer a to b minus how many votes prefer b to a

Solution

- For every a, b , we construct votes π_1, π_2 such that:
 - $\delta_{a,b}(\{\pi_1, \pi_2\}) > 0$;
 - δ for other pairs of candidates is zero
- Let $a = 1, b = 2$ without loss of generality

Reminder

$\delta_{a,b}(\text{voters})$ is how many votes prefer a to b minus how many votes prefer b to a

Solution

- For every a, b , we construct votes π_1, π_2 such that:
 - $\delta_{a,b}(\{\pi_1, \pi_2\}) > 0$;
 - δ for other pairs of candidates is zero
- Let $a = 1, b = 2$ without loss of generality
 - $\pi_1 = (1, 2, 3, 4, \dots, n - 1, n)$

Reminder

$\delta_{a,b}(\text{voters})$ is how many votes prefer a to b minus how many votes prefer b to a

Solution

- For every a, b , we construct votes π_1, π_2 such that:
 - $\delta_{a,b}(\{\pi_1, \pi_2\}) > 0$;
 - δ for other pairs of candidates is zero
- Let $a = 1, b = 2$ without loss of generality
 - $\pi_1 = (1, 2, 3, 4, \dots, n - 1, n)$
 - $\pi_2 = (n, n - 1, \dots, 4, 3, 1, 2)$

Reminder

$\delta_{a,b}(\text{voters})$ is how many voters prefer a to b minus how many voters prefer b to a

Solution

- For every a, b , we construct votes π_1, π_2 such that:
 - $\delta_{a,b}(\{\pi_1, \pi_2\}) > 0$;
 - δ for other pairs of candidates is zero
- Let $a = 1, b = 2$ without loss of generality
 - $\pi_1 = (1, 2, 3, 4, \dots, n - 1, n)$
 - $\pi_2 = (n, n - 1, \dots, 4, 3, 1, 2)$
- Add those voters for each pair of candidates in requirements

Reminder

$\delta_{a,b}(\text{voters})$ is how many voters prefer a to b minus how many voters prefer b to a

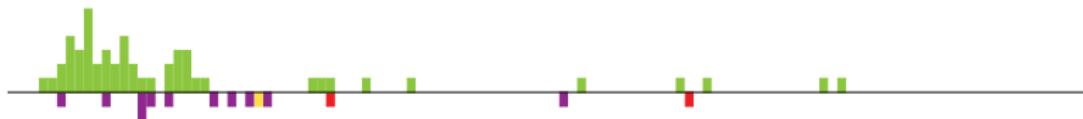
Solution

- For every a, b , we construct votes π_1, π_2 such that:
 - $\delta_{a,b}(\{\pi_1, \pi_2\}) > 0$;
 - δ for other pairs of candidates is zero
- Let $a = 1, b = 2$ without loss of generality
 - $\pi_1 = (1, 2, 3, 4, \dots, n - 1, n)$
 - $\pi_2 = (n, n - 1, \dots, 4, 3, 1, 2)$
- Add those voters for each pair of candidates in requirements
- This uses at most $2\binom{n}{2} \leq n^2$ voters

Number of submissions: 114
of which accepted: 31



First solved by **ETH Zürich** after 40m



The problem

Generate a grid of size at most 2025×2025 with # and . characters which has exactly k rectangles of #, where $k \leq 4 \cdot 10^{12}$ is given

The problem

Generate a grid of size at most 2025×2025 with # and . characters which has exactly k rectangles of #, where $k \leq 4 \cdot 10^{12}$ is given

Solution

- Note that there is not that much leeway: the entire 2025×2025 grid has only $\left(\frac{2025 \cdot 2024}{2}\right)^2 \approx 4.2 \cdot 10^{12}$ rectangles

The problem

Generate a grid of size at most 2025×2025 with # and . characters which has exactly k rectangles of #, where $k \leq 4 \cdot 10^{12}$ is given

Solution

- Note that there is not that much leeway: the entire 2025×2025 grid has only $\left(\frac{2025 \cdot 2024}{2}\right)^2 \approx 4.2 \cdot 10^{12}$ rectangles
- Still, there are many approaches that work, not a single "correct" solution. Here is one approach

The problem

Generate a grid of size at most 2025×2025 with # and . characters which has exactly k rectangles of #, where $k \leq 4 \cdot 10^{12}$ is given

Solution

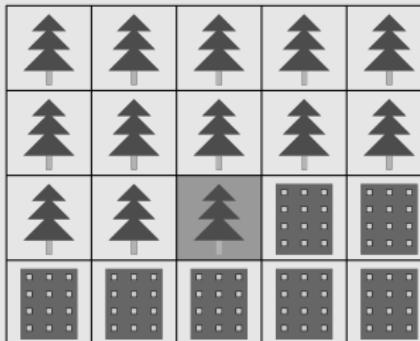
- Note that there is not that much leeway: the entire 2025×2025 grid has only $\left(\frac{2025 \cdot 2024}{2}\right)^2 \approx 4.2 \cdot 10^{12}$ rectangles
- Still, there are many approaches that work, not a single "correct" solution. Here is one approach
- Let us have several connected components. First, we try to have a large one that has close to k rectangles, and then use additional components to cover the difference

Solution

- We start with a 1×2025 rectangle, and then add additional rows. In each row, we add cells one by one, while we still have less than k rectangles

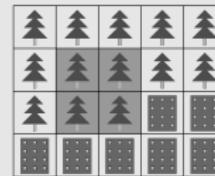
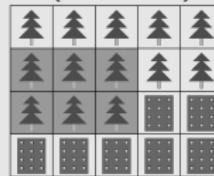
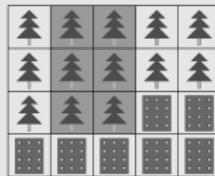
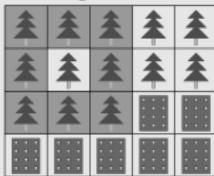
Solution

- We start with a 1×2025 rectangle, and then add additional rows. In each row, we add cells one by one, while we still have less than k rectangles
- (We replace 2025 with 5 for the purpose of the picture, and the last added cell is highlighted)



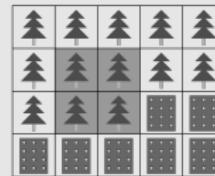
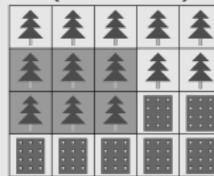
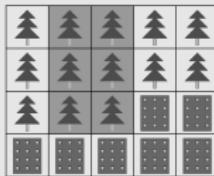
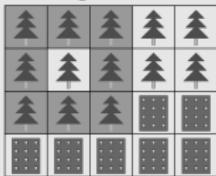
Solution

- Adding a cell in row r and column c (0-based) adds $r \cdot c$ new rectangles:



Solution

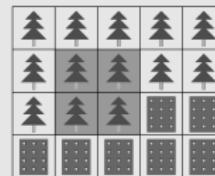
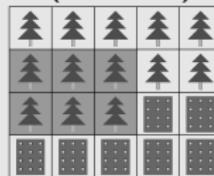
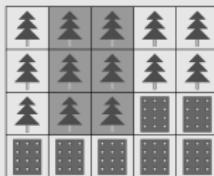
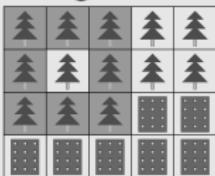
- Adding a cell in row r and column c (0-based) adds $r \cdot c$ new rectangles:



- We reach $4 \cdot 10^{12}$ rectangles within 1977 rows, so we can stop right before that and we will be at most $1976 \cdot 2024 - 1 = 3\,999\,423$ rectangles short of the required k

Solution

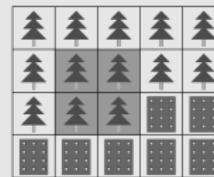
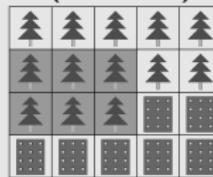
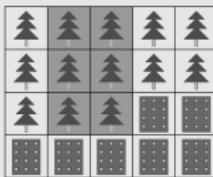
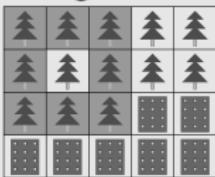
- Adding a cell in row r and column c (0-based) adds $r \cdot c$ new rectangles:



- We reach $4 \cdot 10^{12}$ rectangles within 1977 rows, so we can stop right before that and we will be at most $1976 \cdot 2024 - 1 = 3\,999\,423$ rectangles short of the required k
- Now we can skip one row, and start solving the same problem but with $k \leq 3\,999\,423$

Solution

- Adding a cell in row r and column c (0-based) adds $r \cdot c$ new rectangles:



- We reach $4 \cdot 10^{12}$ rectangles within 1977 rows, so we can stop right before that and we will be at most $1976 \cdot 2024 - 1 = 3\,999\,423$ rectangles short of the required k
- Now we can skip one row, and start solving the same problem but with $k \leq 3\,999\,423$
- We will have one more block with at most 3 rows, and at most six more blocks with two rows, and we're done!

Additional thoughts

- The online mirror has tougher constraints $k \leq 4.194 \cdot 10^{12}$

Additional thoughts

- The online mirror has tougher constraints $k \leq 4.194 \cdot 10^{12}$
- Having independent connected components no longer cuts it, as even a 2024×2024 square has less rectangles — so we need to achieve k in one go

Additional thoughts

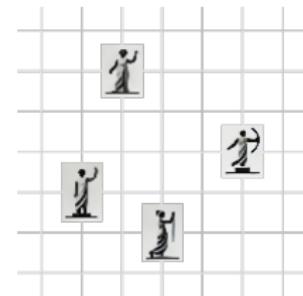
- The online mirror has tougher constraints $k \leq 4.194 \cdot 10^{12}$
- Having independent connected components no longer cuts it, as even a 2024×2024 square has less rectangles — so we need to achieve k in one go
- **Hint:** Consider a 2023×2024 rectangle. What happens if we add some cells to the right of it, and some cells to the bottom of it? What numbers can be represented as $2023x + 2024y$?

Additional thoughts

- The online mirror has tougher constraints $k \leq 4.194 \cdot 10^{12}$
- Having independent connected components no longer cuts it, as even a 2024×2024 square has less rectangles — so we need to achieve k in one go
- **Hint:** Consider a 2023×2024 rectangle. What happens if we add some cells to the right of it, and some cells to the bottom of it? What numbers can be represented as $2023x + 2024y$?
- For the judges' solution, it does not matter if the rectangles are hollow or not. We made them hollow so that counting rectangles takes $\mathcal{O}(n^2 \log n)$ instead of $\mathcal{O}(n^2)$, so hopefully simple hill-climbing solutions were less likely to work

AUTHORED BY: Federico Glaudo & Giovanni Paolini PREPARED BY: Giovanni Paolini

Number of submissions: 61
of which accepted: 15



First solved by **ETH Zürich** after 1h25m



The problem

Given the location of the first and last statue in a 2D city, arrange the remaining $n - 2$ statues so that the Manhattan distance between the i -th and the $(i + 1)$ -th statue is exactly d_i (if possible).

The problem

Given the location of the first and last statue in a 2D city, arrange the remaining $n - 2$ statues so that the Manhattan distance between the i -th and the $(i + 1)$ -th statue is exactly d_i (if possible).

Solution

- Let d_0 be the distance between the first and last statue

The problem

Given the location of the first and last statue in a 2D city, arrange the remaining $n - 2$ statues so that the Manhattan distance between the i -th and the $(i + 1)$ -th statue is exactly d_i (if possible).

Solution

- Let d_0 be the distance between the first and last statue
- **Key insight:** An arrangement exists if and only if
 - (1) $d_0 + d_1 + \dots + d_{n-1}$ is even;
 - (2) $d_i \leq d_0 + \dots + d_{i-1} + d_{i+1} + \dots + d_{n-1}$ for all $0 \leq i \leq n - 1$

The problem

Given the location of the first and last statue in a 2D city, arrange the remaining $n - 2$ statues so that the Manhattan distance between the i -th and the $(i + 1)$ -th statue is exactly d_i (if possible).

Solution

- Let d_0 be the distance between the first and last statue
- **Key insight:** An arrangement exists if and only if
 - (1) $d_0 + d_1 + \dots + d_{n-1}$ is even;
 - (2) $d_i \leq d_0 + \dots + d_{i-1} + d_{i+1} + \dots + d_{n-1}$ for all $0 \leq i \leq n - 1$
- The two conditions are necessary for d_0, d_1, \dots, d_{n-1} to form the (Manhattan) side lengths of a polygon. In particular, (2) follows from the triangle inequality

Solution

■ Conversely, assume that the two conditions hold:

- (1) $d_0 + d_1 + \cdots + d_{n-1}$ is even;
- (2) $d_i \leq d_0 + \cdots + d_{i-1} + d_{i+1} + \cdots + d_{n-1}$ for all $0 \leq i \leq n - 1$.

We inductively show how to construct a valid arrangement

Solution

- Conversely, assume that the two conditions hold:

- (1) $d_0 + d_1 + \cdots + d_{n-1}$ is even;
- (2) $d_i \leq d_0 + \cdots + d_{i-1} + d_{i+1} + \cdots + d_{n-1}$ for all $0 \leq i \leq n - 1$.

We inductively show how to construct a valid arrangement

- Base case:** If $n = 2$, there is no additional statue to place

Solution

- Conversely, assume that the two conditions hold:
 - (1) $d_0 + d_1 + \cdots + d_{n-1}$ is even;
 - (2) $d_i \leq d_0 + \cdots + d_{i-1} + d_{i+1} + \cdots + d_{n-1}$ for all $0 \leq i \leq n - 1$.
- We inductively show how to construct a valid arrangement
 - **Base case:** If $n = 2$, there is no additional statue to place
 - **Induction:** Assuming $n \geq 3$, let s_1, \dots, s_n be the locations of the statues (where s_1 and s_n are given, and the rest must be determined)

Solution

- Conversely, assume that the two conditions hold:

- (1) $d_0 + d_1 + \cdots + d_{n-1}$ is even;
- (2) $d_i \leq d_0 + \cdots + d_{i-1} + d_{i+1} + \cdots + d_{n-1}$ for all $0 \leq i \leq n - 1$.

We inductively show how to construct a valid arrangement

- Base case:** If $n = 2$, there is no additional statue to place
- Induction:** Assuming $n \geq 3$, let s_1, \dots, s_n be the locations of the statues (where s_1 and s_n are given, and the rest must be determined)
- Our aim is to determine s_{n-1} such that:

Solution

- Conversely, assume that the two conditions hold:
 - (1) $d_0 + d_1 + \cdots + d_{n-1}$ is even;
 - (2) $d_i \leq d_0 + \cdots + d_{i-1} + d_{i+1} + \cdots + d_{n-1}$ for all $0 \leq i \leq n - 1$.
- We inductively show how to construct a valid arrangement
- **Base case:** If $n = 2$, there is no additional statue to place
- **Induction:** Assuming $n \geq 3$, let s_1, \dots, s_n be the locations of the statues (where s_1 and s_n are given, and the rest must be determined)
- Our aim is to determine s_{n-1} such that:
 - $\text{distance}(s_{n-1}, s_n) = d_{n-1}$;

Solution

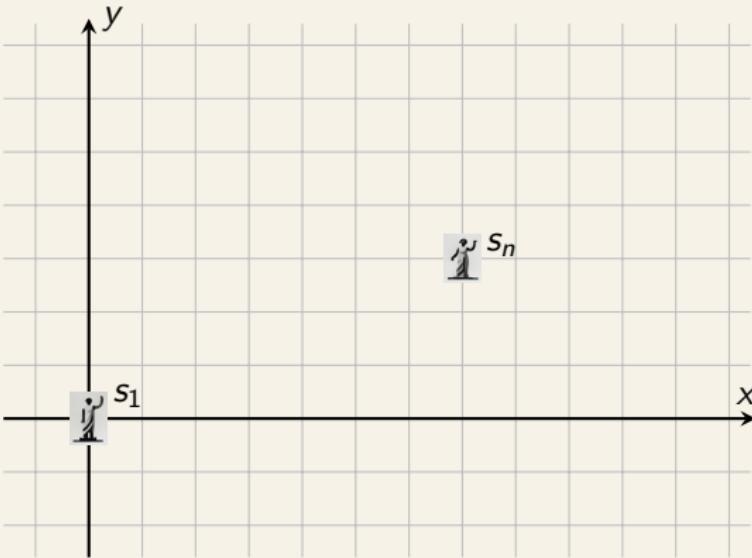
- Conversely, assume that the two conditions hold:

- (1) $d_0 + d_1 + \cdots + d_{n-1}$ is even;
- (2) $d_i \leq d_0 + \cdots + d_{i-1} + d_{i+1} + \cdots + d_{n-1}$ for all $0 \leq i \leq n - 1$.

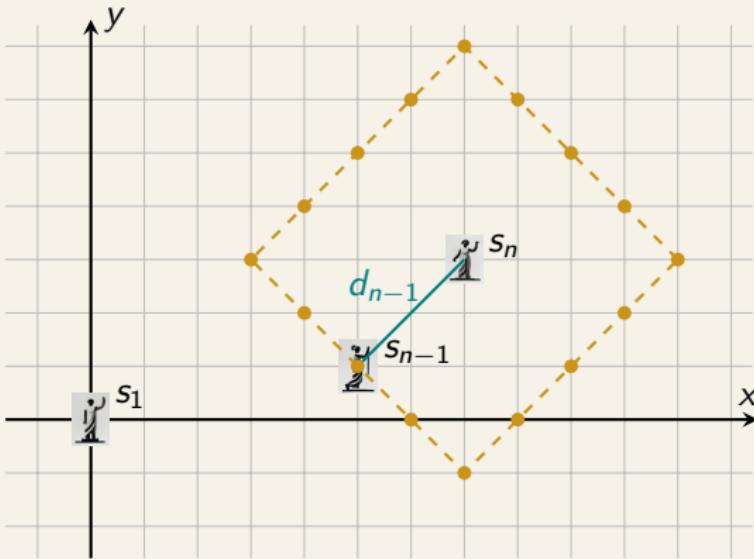
We inductively show how to construct a valid arrangement

- Base case:** If $n = 2$, there is no additional statue to place
- Induction:** Assuming $n \geq 3$, let s_1, \dots, s_n be the locations of the statues (where s_1 and s_n are given, and the rest must be determined)
- Our aim is to determine s_{n-1} such that:
 - distance(s_{n-1}, s_n) = d_{n-1} ;
 - if we define $\hat{d}_0 = \text{distance}(s_1, s_{n-1})$, then the distances $\hat{d}_0, d_1, \dots, d_{n-2}$ satisfy (1) and (2)

Solution

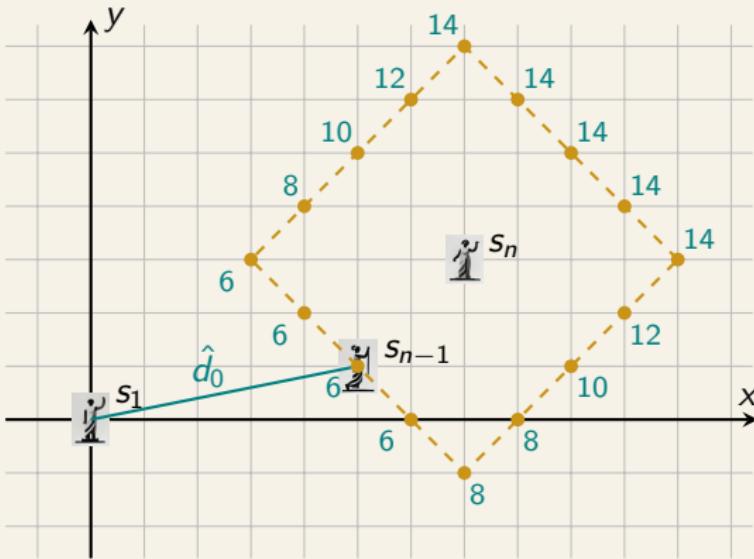


Solution



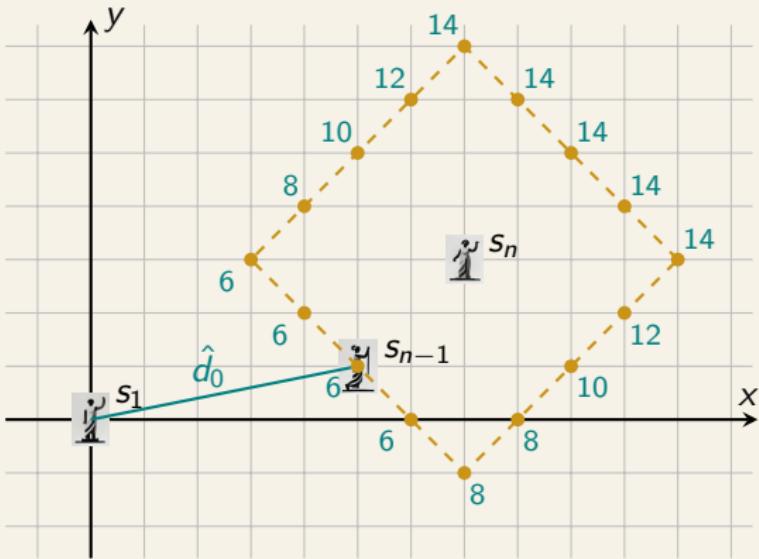
- The statue s_{n-1} must be at distance d_{n-1} from s_n

Solution



- The statue s_{n-1} must be at distance d_{n-1} from s_n
- \hat{d}_0 takes all values between $|d_0 - d_{n-1}|$ and $d_0 + d_{n-1}$ with the same parity as $d_0 + d_{n-1}$

Solution



- The statue s_{n-1} must be at distance d_{n-1} from s_n
- \hat{d}_0 takes all values between $|d_0 - d_{n-1}|$ and $d_0 + d_{n-1}$ with the same parity as $d_0 + d_{n-1}$
- The parity of \hat{d}_0 ensures that $\hat{d}_0, d_1, \dots, d_{n-2}$ satisfy condition (1)

Solution

- **Claim:** The value $\hat{d}_0 := \min(d_0 + d_{n-1}, d_1 + \dots + d_{n-2})$ belongs to the admissible range $\{|d_0 - d_{n-1}|, \dots, d_0 + d_{n-1}\}$, has the correct parity, and makes $\hat{d}_0, d_1, \dots, d_{n-2}$ satisfy condition (2)

Solution

- **Claim:** The value $\hat{d}_0 := \min(d_0 + d_{n-1}, d_1 + \dots + d_{n-2})$ belongs to the admissible range $\{|d_0 - d_{n-1}|, \dots, d_0 + d_{n-1}\}$, has the correct parity, and makes $\hat{d}_0, d_1, \dots, d_{n-2}$ satisfy condition (2)
- This can be easily proved by distinguishing the two cases:
 $\hat{d}_0 = d_0 + d_{n-1}$ and $\hat{d}_0 = d_1 + \dots + d_{n-2}$

Solution

- **Claim:** The value $\hat{d}_0 := \min(d_0 + d_{n-1}, d_1 + \dots + d_{n-2})$ belongs to the admissible range $\{|d_0 - d_{n-1}|, \dots, d_0 + d_{n-1}\}$, has the correct parity, and makes $\hat{d}_0, d_1, \dots, d_{n-2}$ satisfy condition (2)
- This can be easily proved by distinguishing the two cases:
 $\hat{d}_0 = d_0 + d_{n-1}$ and $\hat{d}_0 = d_1 + \dots + d_{n-2}$
- A suitable location for s_{n-1} can then be found in $\mathcal{O}(1)$ time

Solution

- **Claim:** The value $\hat{d}_0 := \min(d_0 + d_{n-1}, d_1 + \dots + d_{n-2})$ belongs to the admissible range $\{|d_0 - d_{n-1}|, \dots, d_0 + d_{n-1}\}$, has the correct parity, and makes $\hat{d}_0, d_1, \dots, d_{n-2}$ satisfy condition (2)
- This can be easily proved by distinguishing the two cases:
 $\hat{d}_0 = d_0 + d_{n-1}$ and $\hat{d}_0 = d_1 + \dots + d_{n-2}$
- A suitable location for s_{n-1} can then be found in $\mathcal{O}(1)$ time
- Overall, we have an $\mathcal{O}(n)$ algorithm to construct a valid arrangement if conditions (1) and (2) are initially satisfied

E

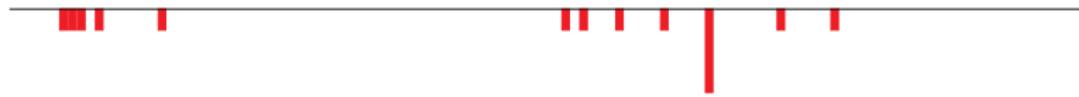
Porto Vs. Benfica

AUTHORED BY: Pedro Paredes PREPARED BY: Pedro Paredes

Number of submissions: 7
of which accepted: 2



First solved by **Scuola Normale Superiore** after 2h21m



The problem

A graph is given and an agent (supporters' club) needs to go from node 1 to node n . An adversary (police), that knows the agent movements, can, at any time and exactly once, remove an edge.

What is the minimum time required to the agent to reach n , if they play optimally?

The problem

A graph is given and an agent (supporters' club) needs to go from node 1 to node n . An adversary (police), that knows the agent movements, can, at any time and exactly once, remove an edge.

What is the minimum time required to the agent to reach n , if they play optimally?

Solution

- Let's call $f(v)$ the minimum number of roads needed to reach n from v

The problem

A graph is given and an agent (supporters' club) needs to go from node 1 to node n . An adversary (police), that knows the agent movements, can, at any time and exactly once, remove an edge.

What is the minimum time required to the agent to reach n , if they play optimally?

Solution

- Let's call $f(v)$ the minimum number of roads needed to reach n from v
- So $f(1)$ is the answer and $f(n)$ is 0

Solution

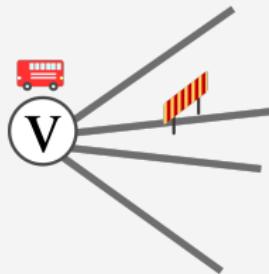
- Suppose no road has been blocked and supporters are in v

Solution

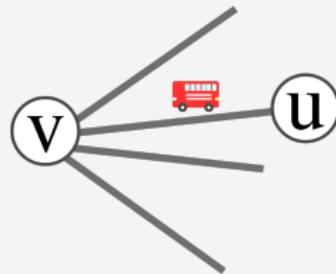
- Suppose no road has been blocked and supporters are in v
- Then the police does one of 2 things:

Solution

- Suppose no road has been blocked and supporters are in v
- Then the police does one of 2 things:



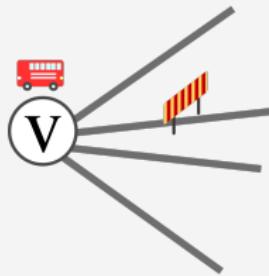
Block some road adjacent to v



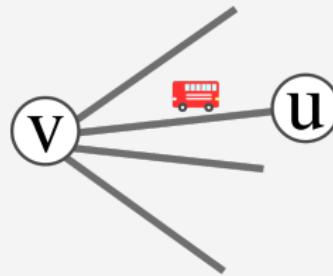
Let the supporters pick some road

Solution

- Suppose no road has been blocked and supporters are in v
- Then the police does one of 2 things:



Cost is $g(v, e) \rightarrow g(v)$



Cost is $f(u) \rightarrow 1 + \min_u(f(u))$

- Let $g(v, e)$ be the shortest path from v to n with edge e blocked, and let $g(v) := \max(g(v, e))$ for adjacent e

Solution

- **Observation 1:** $f(v) = \max(g(v), 1 + \min_{u \sim v}(f(u)))$

Solution

- **Observation 1:** $f(v) = \max(g(v), 1 + \min_{u \sim v}(f(u)))$
- So, now we need to compute g and then f

Solution

- **Observation 1:** $f(v) = \max(g(v), 1 + \min_{u \sim v}(f(u)))$
- So, now we need to compute g and then f
- Let's first find f assuming we know g

AUTHORED BY: Pedro Paredes PREPARED BY: Pedro Paredes

Solution: Computing f assuming g

- We can compute $f(v) = \max(g(v), 1 + \min_{u \sim v}(f(u)))$ using a greedy Dijkstra-like algorithm

Solution: Computing f assuming g

- We can compute $f(v) = \max(g(v), 1 + \min_{u \sim v}(f(u)))$ using a greedy Dijkstra-like algorithm
- Set $f[n] = f(n) = 0$ and initially only vertex n is *processed*

Solution: Computing f assuming g

- We can compute $f(v) = \max(g(v), 1 + \min_{u \sim v}(f(u)))$ using a greedy Dijkstra-like algorithm
- Set $f[n] = f(n) = 0$ and initially only vertex n is *processed*
- At each step find the unprocessed vertex v with the **lowest** $f[v]$, mark it processed, and set:

$$f[v] = \max(g(v), \min_{u \sim v}(1 + f[u]))$$

Solution: Computing f assuming g

- We can compute $f(v) = \max(g(v), 1 + \min_{u \sim v}(f(u)))$ using a greedy Dijkstra-like algorithm
- Set $f[n] = f(n) = 0$ and initially only vertex n is *processed*
- At each step find the unprocessed vertex v with the **lowest** $f[v]$, mark it processed, and set:

$$f[v] = \max(g(v), \min_{u \sim v}(1 + f[u]))$$

- Intuition: applying the formula of f one pair of adjacent vertices at a time

Solution: Computing f assuming g

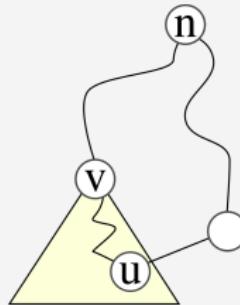
- We can compute $f(v) = \max(g(v), 1 + \min_{u \sim v}(f(u)))$ using a greedy Dijkstra-like algorithm
- Set $f[n] = f(n) = 0$ and initially only vertex n is *processed*
- At each step find the unprocessed vertex v with the **lowest** $f[v]$, mark it processed, and set:

$$f[v] = \max(g(v), \min_{u \sim v}(1 + f[u]))$$

- Intuition: applying the formula of f one pair of adjacent vertices at a time
- **Observation 2:** At the end $f(v) = f[v]$

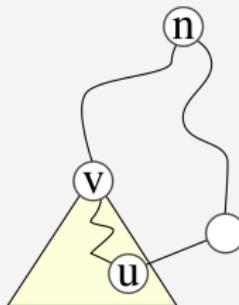
Solution: Computing g

- Consider the BFS tree from n



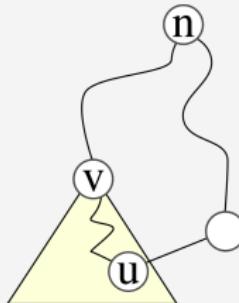
Solution: Computing g

- Consider the BFS tree from n
- Observation 3: $g(v)$ corresponds to:



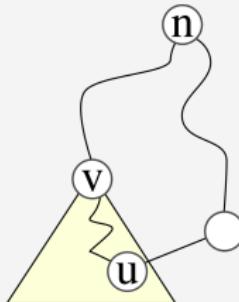
Solution: Computing g

- Consider the BFS tree from n
- Observation 3: $g(v)$ corresponds to:
 1. Start at v



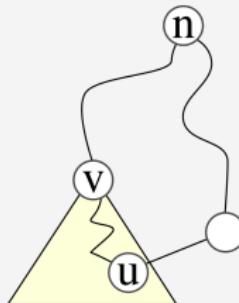
Solution: Computing g

- Consider the BFS tree from n
- Observation 3: $g(v)$ corresponds to:
 1. Start at v
 2. Go down the subtree some number of steps (potentially 0)



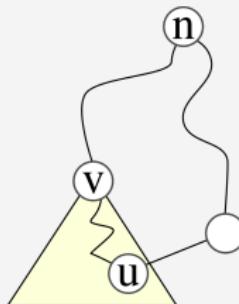
Solution: Computing g

- Consider the BFS tree from n
- Observation 3: $g(v)$ corresponds to:
 1. Start at v
 2. Go down the subtree some number of steps (potentially 0)
 3. Take one non-tree edge that outside v 's subtree



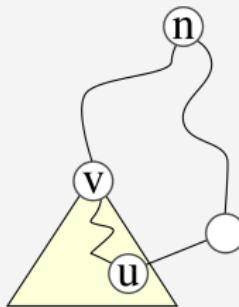
Solution: Computing g

- Consider the BFS tree from n
- Observation 3: $g(v)$ corresponds to:
 1. Start at v
 2. Go down the subtree some number of steps (potentially 0)
 3. Take one non-tree edge that outside v 's subtree
 4. Take the shortest path from that vertex to n



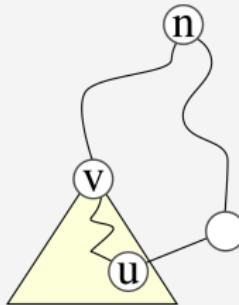
Solution: Computing g

- Find all valid paths from each v



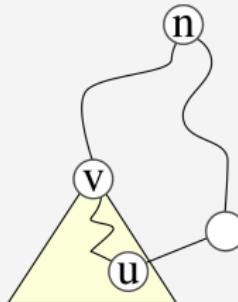
Solution: Computing g

- Find all valid paths from each v
- Recursively compute distances for all children of v



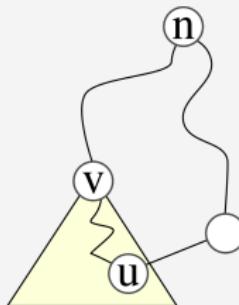
Solution: Computing g

- Find all valid paths from each v
- Recursively compute distances for all children of v
- Initialize sets by adding pairs $(1 + \text{dist}(u), u)$ for each non-tree edge



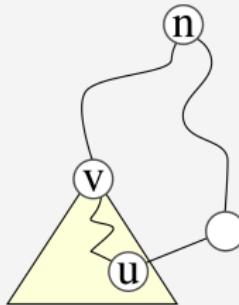
Solution: Computing g

- Merge v 's set with all its children, adding 1 to the distance values



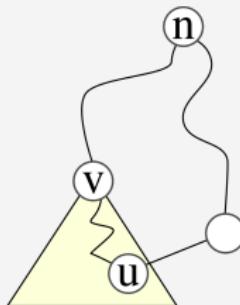
Solution: Computing g

- Merge v 's set with all its children, adding 1 to the distance values
- If minimum element (d, u) belongs to v 's subtree, remove it



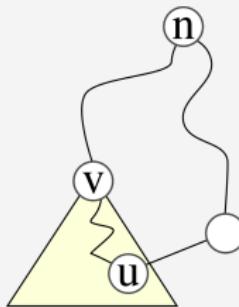
Solution: Computing g

- Merge v 's set with all its children, adding 1 to the distance values
- If minimum element (d, u) belongs to v 's subtree, remove it
- The remaining minimum element in the set gives $g(v)$



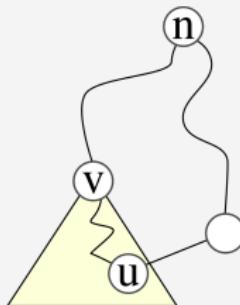
Solution: Computing g

- Use DSU (union-find) to check if u is in vs subtree



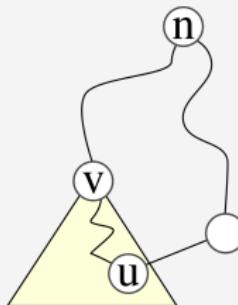
Solution: Computing g

- Use DSU (union-find) to check if u is in vs subtree
- Use small-to-large merging to efficiently merge sets



Solution: Computing g

- Use DSU (union-find) to check if u is in vs subtree
- Use small-to-large merging to efficiently merge sets
- Total running time of this step is $\mathcal{O}(n \log^2 m)$



K

Amusement Park Rides

AUTHORED BY: Egor Kulikov PREPARED BY: Egor Kulikov

Number of submissions: 33
of which accepted: 2



First solved by **Jagiellonian University in Krakow** after 2h27m



The problem

We are given a sequence of n numbers a_i . We need to find another sequence of n *distinct* numbers b_i such that:

- $a_i \mid b_i$ for each $1 \leq i \leq n$;
- $\max(b_i)$ is the minimum possible.

(Output only the minimum)

The problem

We are given a sequence of n numbers a_i . We need to find another sequence of n *distinct* numbers b_i such that:

- $a_i \mid b_i$ for each $1 \leq i \leq n$;
- $\max(b_i)$ is the minimum possible.

(Output only the minimum)

Solution

- **Strategy:** Build a graph and run the max flow algorithm

AUTHORED BY: Egor Kulikov PREPARED BY: Egor Kulikov

The problem

We are given a sequence of n numbers a_i . We need to find another sequence of n *distinct* numbers b_i such that:

- $a_i \mid b_i$ for each $1 \leq i \leq n$;
- $\max(b_i)$ is the minimum possible.

(Output only the minimum)

Solution

- **Strategy:** Build a graph and run the max flow algorithm
- When we reach flow n , the last vertex added will determine the answer

The problem

We are given a sequence of n numbers a_i . We need to find another sequence of n *distinct* numbers b_i such that:

- $a_i \mid b_i$ for each $1 \leq i \leq n$;
- $\max(b_i)$ is the minimum possible.

(Output only the minimum)

Solution

- **Strategy:** Build a graph and run the max flow algorithm
- When we reach flow n , the last vertex added will determine the answer
- Denote p_1, \dots, p_k the distinct values that appear in the sequence, where p_j appears q_j times

AUTHORED BY: Egor Kulikov PREPARED BY: Egor Kulikov

Solution: Graph structure

- **Vertices:** The initial set of vertices is made up of:
 - a source S and a sink T ;
 - k other vertices A_1, \dots, A_k

Solution: Graph structure

- **Vertices:** The initial set of vertices is made up of:
 - a source S and a sink T ;
 - k other vertices A_1, \dots, A_k
- **Edges:** Initially, create edges from S to each A_i with capacity q_i

Solution: Graph structure

- **Vertices:** The initial set of vertices is made up of:
 - a source S and a sink T ;
 - k other vertices A_1, \dots, A_k
- **Edges:** Initially, create edges from S to each A_i with capacity q_i
- **Evolution:** For every positive integer i , add a vertex B_i and:

Solution: Graph structure

- **Vertices:** The initial set of vertices is made up of:
 - a source S and a sink T ;
 - k other vertices A_1, \dots, A_k
- **Edges:** Initially, create edges from S to each A_i with capacity q_i
- **Evolution:** For every positive integer i , add a vertex B_i and:
 - for each A_j with $1 \leq j \leq k$ such that $i \equiv 0 \pmod{p_j}$, add an edge $A_j \xrightarrow{1} B_i$;

Solution: Graph structure

- **Vertices:** The initial set of vertices is made up of:
 - a source S and a sink T ;
 - k other vertices A_1, \dots, A_k
- **Edges:** Initially, create edges from S to each A_i with capacity q_i
- **Evolution:** For every positive integer i , add a vertex B_i and:
 - for each A_j with $1 \leq j \leq k$ such that $i \equiv 0 \pmod{p_j}$, add an edge $A_j \xrightarrow{1} B_i$;
 - add an edge $B_i \xrightarrow{1} T$

Solution: Graph structure

- **Vertices:** The initial set of vertices is made up of:
 - a source S and a sink T ;
 - k other vertices A_1, \dots, A_k
- **Edges:** Initially, create edges from S to each A_i with capacity q_i
- **Evolution:** For every positive integer i , add a vertex B_i and:
 - for each A_j with $1 \leq j \leq k$ such that $i \equiv 0 \pmod{p_j}$, add an edge $A_j \xrightarrow{1} B_i$;
 - add an edge $B_i \xrightarrow{1} T$
- Continue adding vertices B_i in increasing order of i until the (S, T) max flow reaches n . The answer to the problem is the last index i added

AUTHORED BY: Egor Kulikov PREPARED BY: Egor Kulikov

Solution: Implementation

- If we add a vertex B_i for every i naively, the above solution runs in time $\mathcal{O}(\text{ans} \cdot E)$ (where ans is the max flow and E is the number of edges) which is too slow

Solution: Implementation

- If we add a vertex B_i for every i naively, the above solution runs in time $\mathcal{O}(\text{ans} \cdot E)$ (where ans is the max flow and E is the number of edges) which is too slow
- Instead, we maintain a mapping

$$M : \mathbb{Z}^+ \rightarrow \mathcal{P}(\{1, \dots, k\}),$$

where $M(i)$ is a set of indices

Solution: Implementation

- If we add a vertex B_i for every i naively, the above solution runs in time $\mathcal{O}(\text{ans} \cdot E)$ (where ans is the max flow and E is the number of edges) which is too slow
- Instead, we maintain a mapping

$$M : \mathbb{Z}^+ \rightarrow \mathcal{P}(\{1, \dots, k\}),$$

where $M(i)$ is a set of indices

- Initially, set $M(p_j) = \{j\}$ for $1 \leq j \leq k$, and $M(i) = \emptyset$ otherwise

AUTHORED BY: Egor Kulikov PREPARED BY: Egor Kulikov

Solution: Implementation

- At each step, find the smallest i such that $M(i) \neq \emptyset$

AUTHORED BY: Egor Kulikov PREPARED BY: Egor Kulikov

Solution: Implementation

- At each step, find the smallest i such that $M(i) \neq \emptyset$
- Add the B_i along with its edge to T and add edges $A_j \rightarrow B_i$ for every $j \in M(i)$

Solution: Implementation

- At each step, find the smallest i such that $M(i) \neq \emptyset$
- Add the B_i along with its edge to T and add edges $A_j \rightarrow B_i$ for every $j \in M(i)$
- **If** adding these edges increases the maximum flow, then for each $j \in M(i)$ add j to $M(i + p_j)$; **otherwise**, nothing else is needed

Solution: Implementation

- At each step, find the smallest i such that $M(i) \neq \emptyset$
- Add the B_i along with its edge to T and add edges $A_j \rightarrow B_i$ for every $j \in M(i)$
- If adding these edges increases the maximum flow, then for each $j \in M(i)$ add j to $M(i + p_j)$; otherwise, nothing else is needed
- Finally, clear $M(i)$

Solution: Implementation

- At each step, find the smallest i such that $M(i) \neq \emptyset$
- Add the B_i along with its edge to T and add edges $A_j \rightarrow B_i$ for every $j \in M(i)$
- If adding these edges increases the maximum flow, then for each $j \in M(i)$ add j to $M(i + p_j)$; otherwise, nothing else is needed
- Finally, clear $M(i)$
- Complexity:

Solution: Implementation

- At each step, find the smallest i such that $M(i) \neq \emptyset$
- Add the B_i along with its edge to T and add edges $A_j \rightarrow B_i$ for every $j \in M(i)$
- If adding these edges increases the maximum flow, then for each $j \in M(i)$ add j to $M(i + p_j)$; otherwise, nothing else is needed
- Finally, clear $M(i)$
- Complexity:
 - At most $n + k$ distinct values of i are examined because each step either increases the maximum flow or removes at least one element from $\bigcup_i M(i)$

Solution: Implementation

- At each step, find the smallest i such that $M(i) \neq \emptyset$
- Add the B_i along with its edge to T and add edges $A_j \rightarrow B_i$ for every $j \in M(i)$
- If adding these edges increases the maximum flow, then for each $j \in M(i)$ add j to $M(i + p_j)$; otherwise, nothing else is needed
- Finally, clear $M(i)$
- Complexity:
 - At most $n + k$ distinct values of i are examined because each step either increases the maximum flow or removes at least one element from $\bigcup_i M(i)$
 - Overall, the optimized solution runs in $\mathcal{O}(n \cdot E)$ time

Solution: Implementation

- At each step, find the smallest i such that $M(i) \neq \emptyset$
- Add the B_i along with its edge to T and add edges $A_j \rightarrow B_i$ for every $j \in M(i)$
- If adding these edges increases the maximum flow, then for each $j \in M(i)$ add j to $M(i + p_j)$; otherwise, nothing else is needed
- Finally, clear $M(i)$
- Complexity:
 - At most $n + k$ distinct values of i are examined because each step either increases the maximum flow or removes at least one element from $\bigcup_i M(i)$
 - Overall, the optimized solution runs in $\mathcal{O}(n \cdot E)$ time
 - It can be shown that $E = \mathcal{O}(n \log n)$, making the method efficient

AUTHORED BY: Egor Kulikov PREPARED BY: Egor Kulikov

Key points

- We construct a flow graph with vertices $\{A_i\}$ and $\{B_i\}$ to model the problem

Key points

- We construct a flow graph with vertices $\{A_i\}$ and $\{B_i\}$ to model the problem
- The naive method involves adding every B_i until the max flow reaches n , which is slow

Key points

- We construct a flow graph with vertices $\{A_i\}$ and $\{B_i\}$ to model the problem
- The naive method involves adding every B_i until the max flow reaches n , which is slow
- An optimized approach uses a map $M(i)$ to only process necessary indices, reducing the number of steps

Key points

- We construct a flow graph with vertices $\{A_i\}$ and $\{B_i\}$ to model the problem
- The naive method involves adding every B_i until the max flow reaches n , which is slow
- An optimized approach uses a map $M(i)$ to only process necessary indices, reducing the number of steps
- The final complexity is $\mathcal{O}(n \cdot E)$ with $E = \mathcal{O}(n \log n)$

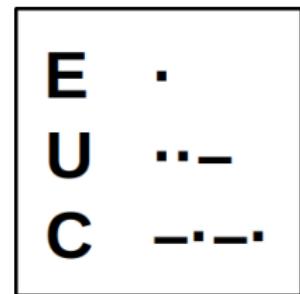
D

Morse Code

AUTHORED BY: Jorke De Vlas

PREPARED BY: Jorke De Vlas

Number of submissions: 10
of which accepted: 1



First solved by **Karlsruhe Institute of Technology** after 3h31m



The problem

Given a set of character frequencies, determine a Morse Code like assignment where no codes are prefixes of each other and where the expected transmission time per character is minimal.

A	$f_A = 0.3$
B	$f_B = 0.6$
C	$f_C = 0.1$

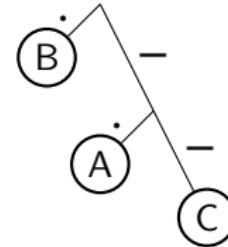
The problem

Given a set of character frequencies, determine a Morse Code like assignment where no codes are prefixes of each other and where the expected transmission time per character is minimal.

Solution

- We can describe such an assignment using a binary tree

A	$f_A = 0.3$
B	$f_B = 0.6$
C	$f_C = 0.1$



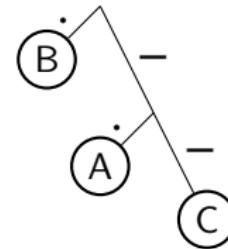
The problem

Given a set of character frequencies, determine a Morse Code like assignment where no codes are prefixes of each other and where the expected transmission time per character is minimal.

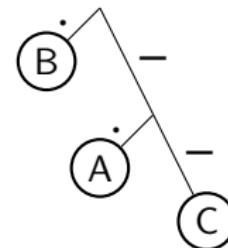
Solution

- We can describe such an assignment using a binary tree
- Only place characters on leaves to avoid prefixes

A	$f_A = 0.3$
B	$f_B = 0.6$
C	$f_C = 0.1$



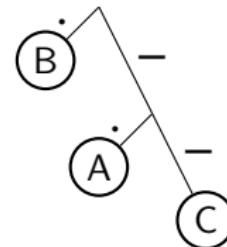
A	$f_A = 0.3$
B	$f_B = 0.6$
C	$f_C = 0.1$



Solution

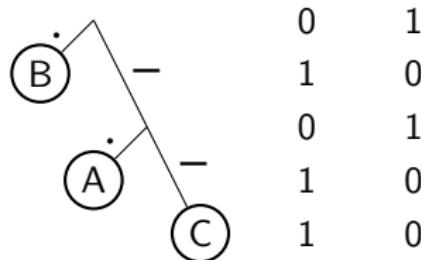
- If we know the shape of the tree, we can assign characters greedily

A	$f_A = 0.3$
B	$f_B = 0.6$
C	$f_C = 0.1$



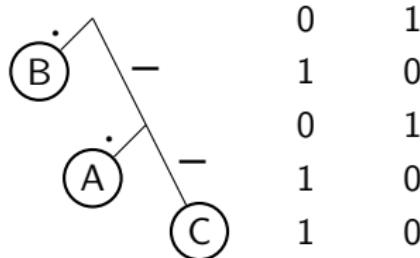
Solution

- If we know the shape of the tree, we can assign characters greedily
- To determine the shape, we only need to know the number of leaves and internal vertices per level



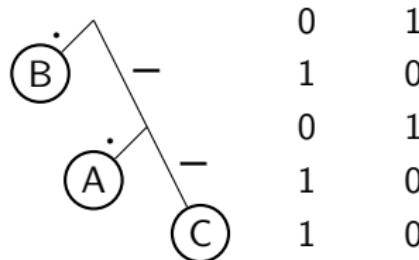
Solution

- The optimal vertex sequences can be determined by trying all possible suffixes with **Dynamic Programming**



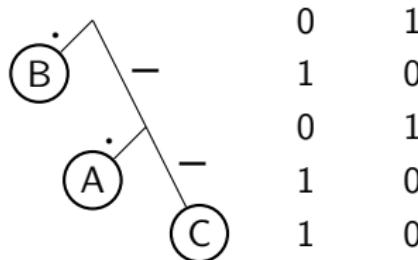
Solution

- The optimal vertex sequences can be determined by trying all possible suffixes with **Dynamic Programming**
- We need three parameters to describe a state:



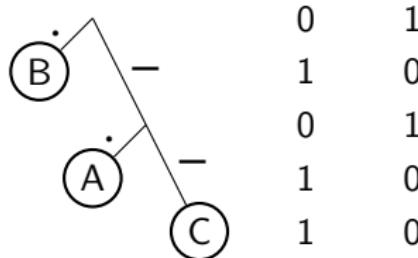
Solution

- The optimal vertex sequences can be determined by trying all possible suffixes with **Dynamic Programming**
- We need three parameters to describe a state:
 - The number v_c of “free” vertices at the current level



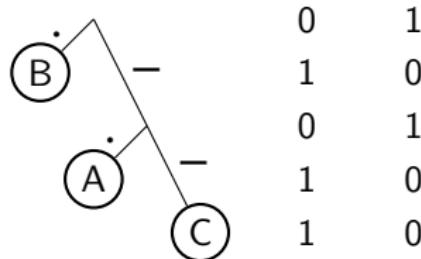
Solution

- The optimal vertex sequences can be determined by trying all possible suffixes with **Dynamic Programming**
- We need three parameters to describe a state:
 - The number v_c of “free” vertices at the current level
 - The number v_n of “free” vertices at the next level



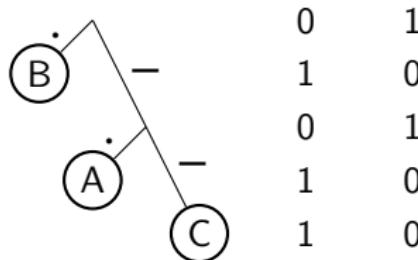
Solution

- The optimal vertex sequences can be determined by trying all possible suffixes with **Dynamic Programming**
- We need three parameters to describe a state:
 - The number v_c of “free” vertices at the current level
 - The number v_n of “free” vertices at the next level
 - The number a of leafs at the current level and below



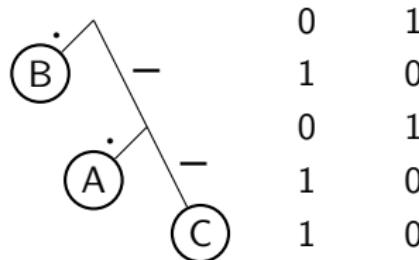
Solution

- The value $dp(v_c, v_n, a)$ describes the expected cost of all characters below the current level



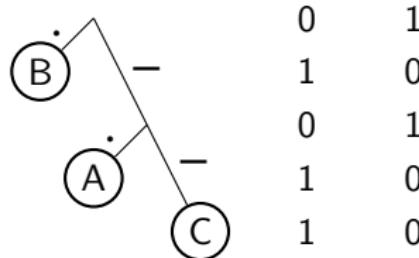
Solution

- The value $dp(v_c, v_n, a)$ describes the expected cost of all characters below the current level
- Sought value:** $dp(1, 0, n)$



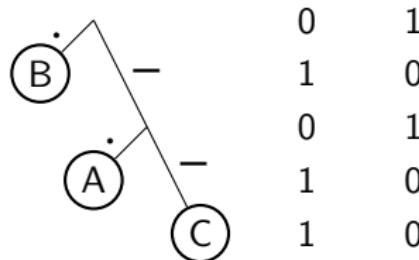
Solution

- The value $dp(v_c, v_n, a)$ describes the expected cost of all characters below the current level
- Sought value:** $dp(1, 0, n)$
- Base case:** $dp(0, 0, 0) = 0$



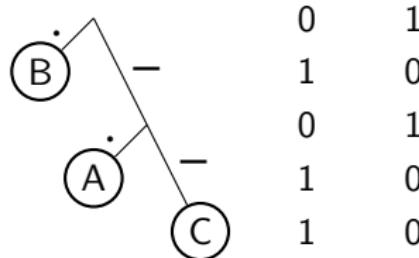
Solution

- The value $dp(v_c, v_n, a)$ describes the expected cost of all characters below the current level
- Sought value:** $dp(1, 0, n)$
- Base case:** $dp(0, 0, 0) = 0$
- Boundary case:** $dp(0, 0, k) = \infty$



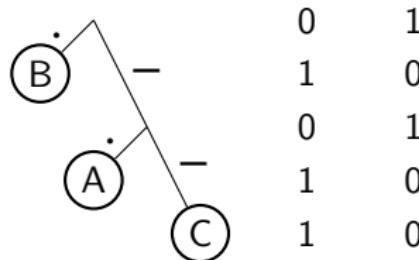
Solution

- The value $dp(v_c, v_n, a)$ describes the expected cost of all characters below the current level
- Sought value:** $dp(1, 0, n)$
- Base case:** $dp(0, 0, 0) = 0$
- Boundary case:** $dp(0, 0, k) = \infty$
- Suboptimal case:** If $v_c + v_n > a$, then $dp(v_c, v_n, a) = \infty$



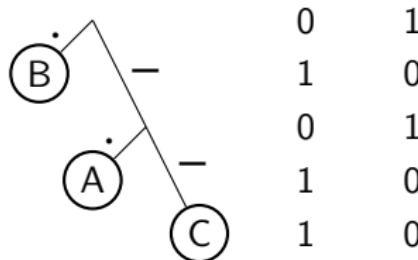
Solution

- In all other cases, we need a transition function. This is the minimum of two options:



Solution

- In all other cases, we need a transition function. This is the minimum of two options:
 - We turn a free vertex into a leaf:
$$\text{dp}(v_c, v_n, a) = \text{dp}(v_c - 1, v_n, a - 1)$$
(only available if $v_c, a > 0$)



Solution

- In all other cases, we need a transition function. This is the minimum of two options:
 - We turn a free vertex into a leaf:
$$\text{dp}(v_c, v_n, a) = \text{dp}(v_c - 1, v_n, a - 1)$$
 (only available if $v_c, a > 0$)
 - We do not want more leaves, and go to the next level:
$$\text{dp}(v_c, v_n, a) = \text{dp}(v_n + v_c, v_c, a) + F(a)$$
, where $F(a)$ is the sum of the a smallest frequencies

Solution

- If you precompute F , the transition function becomes $\mathcal{O}(1)$

Solution

- If you precompute F , the transition function becomes $\mathcal{O}(1)$
- Each state is bounded by v_c , $v_n \geq 0$ and $v_c + v_n \leq a \leq n$, so there are $\mathcal{O}(n^3)$ states

Solution

- If you precompute F , the transition function becomes $\mathcal{O}(1)$
- Each state is bounded by v_c , $v_n \geq 0$ and $v_c + v_n \leq a \leq n$, so there are $\mathcal{O}(n^3)$ states
- This gives a runtime of $\mathcal{O}(n^3)$

Solution

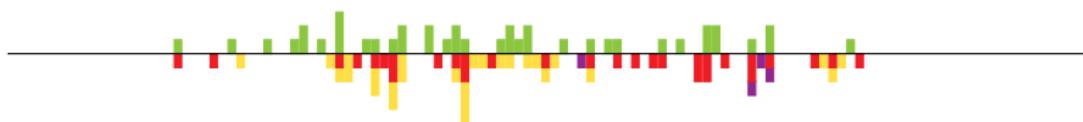
- If you precompute F , the transition function becomes $\mathcal{O}(1)$
- Each state is bounded by $v_c, v_n \geq 0$ and $v_c + v_n \leq a \leq n$, so there are $\mathcal{O}(n^3)$ states
- This gives a runtime of $\mathcal{O}(n^3)$
- To reconstruct the codes, use backtracking, keeping track of a set of partial codes at the current and the next level

AUTHORED BY: Federico Glaudo PREPARED BY: Federico Glaudo

Number of submissions: 6
of which accepted: 1



First solved by **Delft University of Technology** after 3h34m



The problem

You are given an infinite grid where each cell contains a number and the $n \times n$ pattern of numbers is repeated periodically. You can move from a cell with number x to an adjacent cell with number y in $|x - y| + 1$ seconds. How many cells can you reach, approximately, in $R = 10^{20}$ seconds?

The problem

You are given an infinite grid where each cell contains a number and the $n \times n$ pattern of numbers is repeated periodically. You can move from a cell with number x to an adjacent cell with number y in $|x - y| + 1$ seconds. How many cells can you reach, approximately, in $R = 10^{20}$ seconds?

Solution

- **Definition:** We say that a path is *interesting* if:

The problem

You are given an infinite grid where each cell contains a number and the $n \times n$ pattern of numbers is repeated periodically. You can move from a cell with number x to an adjacent cell with number y in $|x - y| + 1$ seconds. How many cells can you reach, approximately, in $R = 10^{20}$ seconds?

Solution

- **Definition:** We say that a path is *interesting* if:
 - the initial and final cell are equivalent modulo n ;

The problem

You are given an infinite grid where each cell contains a number and the $n \times n$ pattern of numbers is repeated periodically. You can move from a cell with number x to an adjacent cell with number y in $|x - y| + 1$ seconds. How many cells can you reach, approximately, in $R = 10^{20}$ seconds?

Solution

■ **Definition:** We say that a path is *interesting* if:

- the initial and final cell are equivalent modulo n ;
- its length is $\leq n^2$

The problem

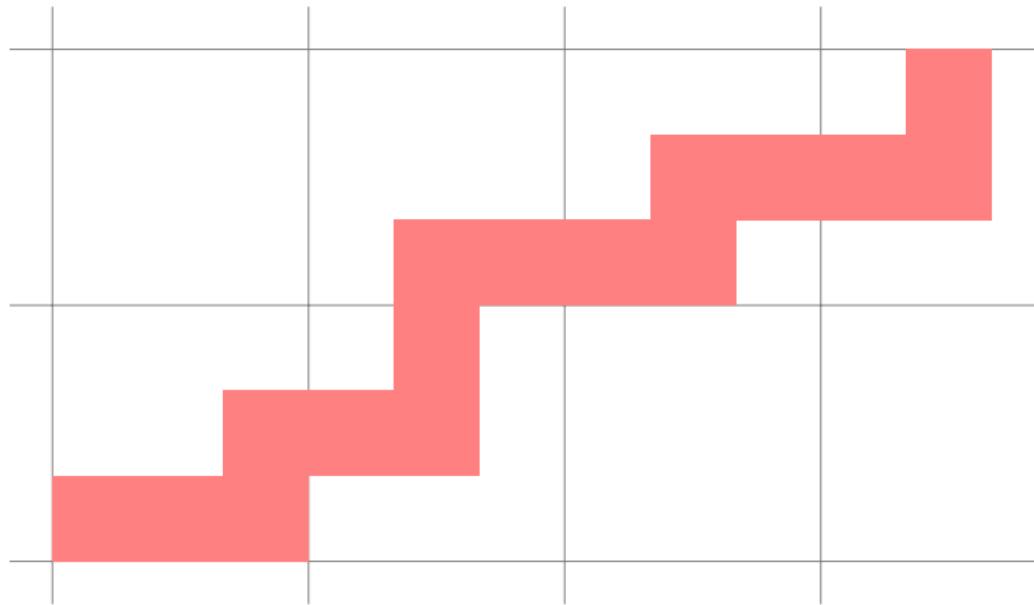
You are given an infinite grid where each cell contains a number and the $n \times n$ pattern of numbers is repeated periodically. You can move from a cell with number x to an adjacent cell with number y in $|x - y| + 1$ seconds. How many cells can you reach, approximately, in $R = 10^{20}$ seconds?

Solution

- **Definition:** We say that a path is *interesting* if:
 - the initial and final cell are equivalent modulo n ;
 - its length is $\leq n^2$
- **Main idea:** Any path can be *split* in **the sum of many interesting paths** plus a remainder of at most n^2 cells

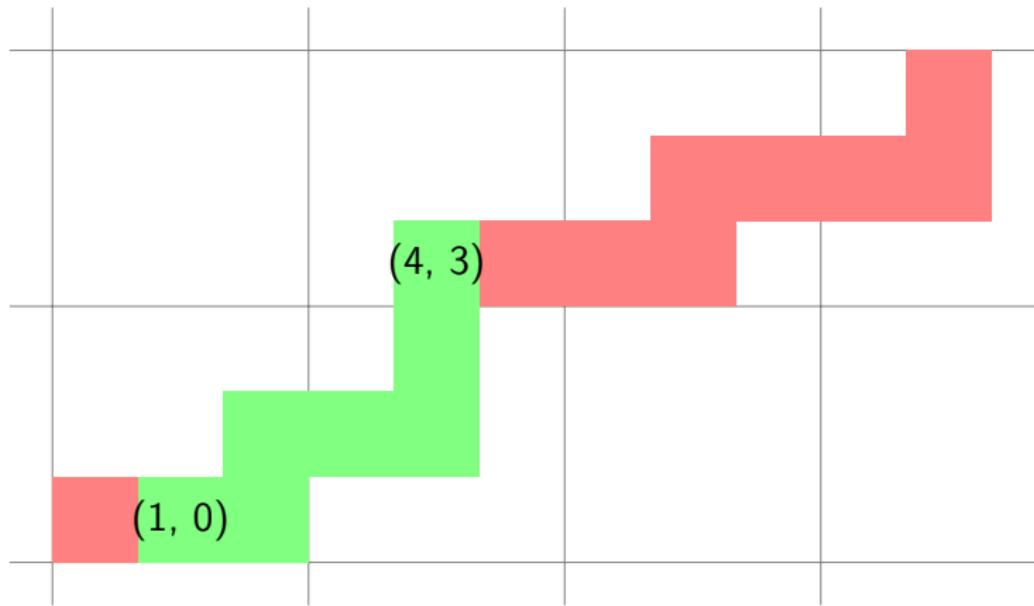
AUTHORED BY: Federico Glaudo PREPARED BY: Federico Glaudo

$$n = 3$$



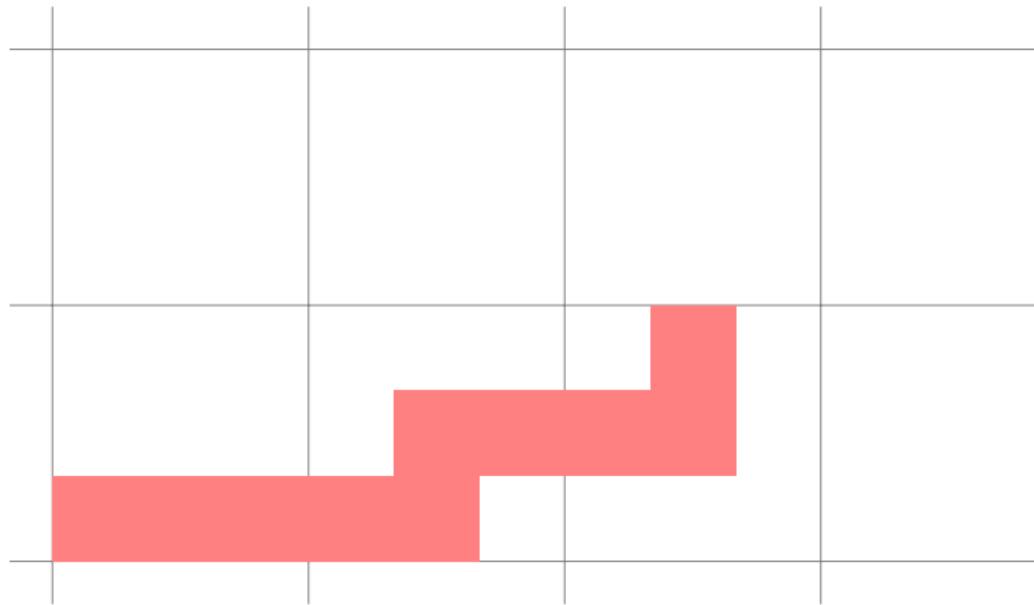
AUTHORED BY: Federico Glaudo PREPARED BY: Federico Glaudo

$$n = 3$$



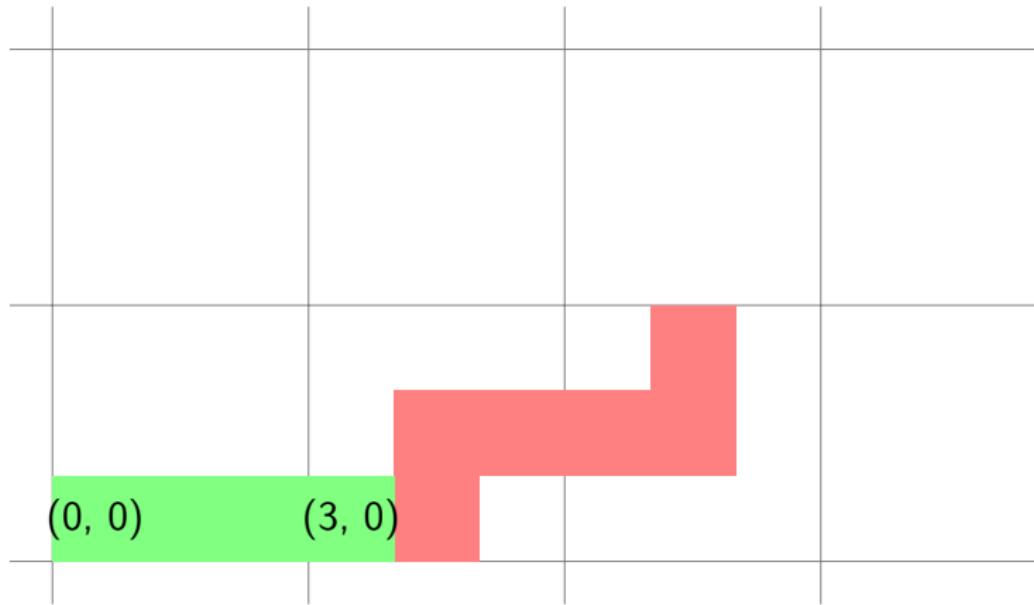
AUTHORED BY: Federico Glaudo PREPARED BY: Federico Glaudo

$$n = 3$$

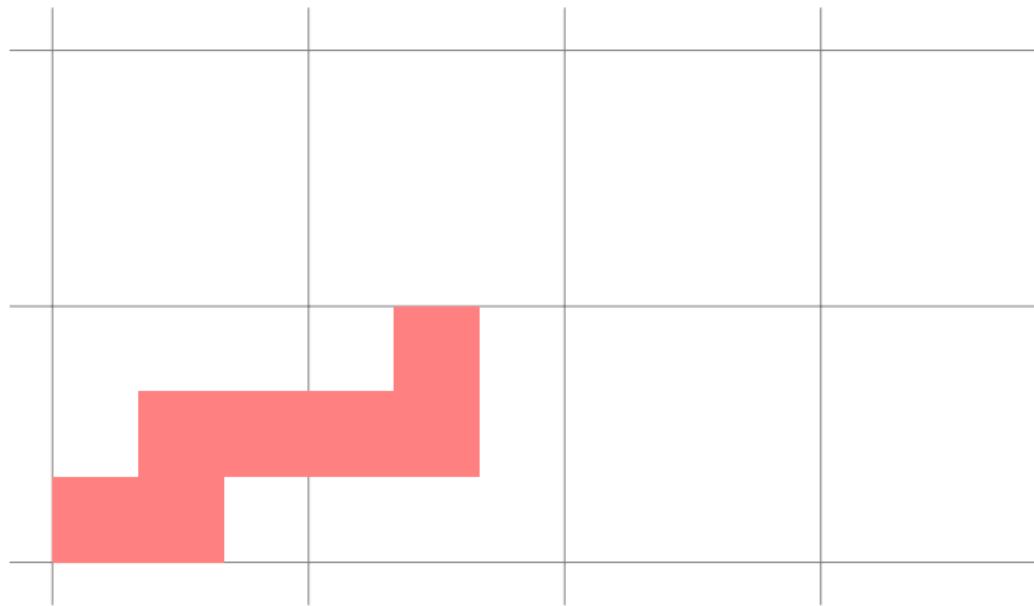


AUTHORED BY: Federico Glaudo PREPARED BY: Federico Glaudo

$$n = 3$$



$$n = 3$$



The problem

You are given an infinite grid where each cell contains a number and the $n \times n$ pattern of numbers is repeated periodically. You can move from a cell with number x to an adjacent cell with number y in $|x - y| + 1$ seconds. How many cells can you reach, approximately, in $R = 10^{20}$ seconds?

Solution

- **Definition:** We say that a path is *interesting* if
 - the initial and final cell are equivalent modulo n ;
 - its length is $\leq n^2$
- **Main idea:** Any path can be *split* in **the sum of many interesting paths** plus a remainder of at most n^2 cells

The problem

You are given an infinite grid where each cell contains a number and the $n \times n$ pattern of numbers is repeated periodically. You can move from a cell with number x to an adjacent cell with number y in $|x - y| + 1$ seconds. How many cells can you reach, approximately, in $R = 10^{20}$ seconds?

Solution

- **Definition:** We say that a path is *interesting* if
 - the initial and final cell are equivalent modulo n ;
 - its length is $\leq n^2$
- **Main idea:** Any path can be *split* in **the sum of many interesting paths** plus a remainder of at most n^2 cells
- **Main idea (“converse”):** Any sum of interesting paths can be *obtained* from a given path up to adding at most $\text{poly}(n)$ cells

Subproblem equivalent to the original one

For a given (a, b) , for any i, j with $|i| + |j| \leq n$, $T(i, j)$ is the minimum distance from (a, b) to $(a + in, b + jn)$ divided by n .

We start from $(0, 0)$ and we can jump to (i, j) in $T(i, j)$ seconds.
How many cells can we reach in $R = 10^{20}$ seconds?

Subproblem equivalent to the original one

For a given (a, b) , for any i, j with $|i| + |j| \leq n$, $T(i, j)$ is the minimum distance from (a, b) to $(a + in, b + jn)$ divided by n . We start from $(0, 0)$ and we can jump to (i, j) in $T(i, j)$ seconds. How many cells can we reach in $R = 10^{20}$ seconds?

Solution: Solving the subproblem

If a, b are large,

$$\text{dist}((0, 0), (a, b)) \approx \inf \left\{ \sum_{|i|+|j|\leq n} \lambda_{ij} T(i, j) : \sum_{|i|+|j|} \lambda_{ij}(i, j) = (x, y) \right\}$$

Subproblem equivalent to the original one

For a given (a, b) , for any i, j with $|i| + |j| \leq n$, $T(i, j)$ is the minimum distance from (a, b) to $(a + in, b + jn)$ divided by n . We start from $(0, 0)$ and we can jump to (i, j) in $T(i, j)$ seconds. How many cells can we reach in $R = 10^{20}$ seconds?

Solution: Solving the subproblem

The answer to the subproblem is

$$\text{area}(P) \cdot R^2,$$

where P is the convex hull of the points $p_{ij} := \left(\frac{i}{T(i,j)}, \frac{j}{T(i,j)} \right)$

Solution: Computing the answer

- It remains only to compute $T(i, j)$ for all $|i| + |j| \leq n$

Solution: Computing the answer

- It remains only to compute $T(i, j)$ for all $|i| + |j| \leq n$
- **The naive way:** Iterate over all initial points (a, b) and all final points $(a + in, b + jn)$. Running Dijkstra requires $O(n^4 \log(n))$ operations. The total runtime is $\mathcal{O}(n^8 \log(n))$, which is too slow

Solution: Computing the answer

- It remains only to compute $T(i, j)$ for all $|i| + |j| \leq n$
- **The naive way:** Iterate over all initial points (a, b) and all final points $(a + in, b + jn)$. Running Dijkstra requires $\mathcal{O}(n^4 \log(n))$ operations. The total runtime is $\mathcal{O}(n^8 \log(n))$, which is too slow
- **The clever way:** Iterate only over the initial points (a, b) . The total runtime is $\mathcal{O}(n^6 \log(n))$, which is enough!

Solution: Computing the answer

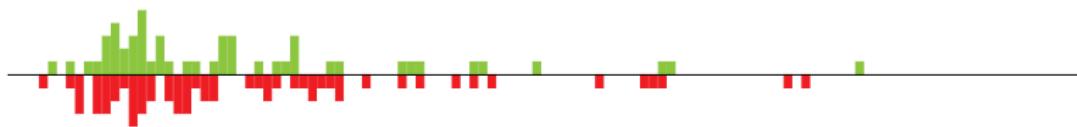
- It remains only to compute $T(i, j)$ for all $|i| + |j| \leq n$
- **The naive way:** Iterate over all initial points (a, b) and all final points $(a + in, b + jn)$. Running Dijkstra requires $\mathcal{O}(n^4 \log(n))$ operations. The total runtime is $\mathcal{O}(n^8 \log(n))$, which is too slow
- **The clever way:** Iterate only over the initial points (a, b) . The total runtime is $\mathcal{O}(n^6 \log(n))$, which is enough!
- **The clever-er way:** Any path that begins and ends in cells equivalent modulo n passes through a cell with one of the two coordinates divisible by n . Iterate only over the initial points (a, b) with $a = 0$ or $b = 0$ and implement a linear-time Dijkstra. The total runtime is $\mathcal{O}(n^5)$

AUTHORED BY: Lucian Bicsi PREPARED BY: Lucian Bicsi

Number of submissions: 0
of which accepted: 0



First solved by N/A after N/A



I Pinball

AUTHORED BY: Lucian Bicsi PREPARED BY: Lucian Bicsi

The problem

You are playing a game on a $h \times w$ grid consisting of free cells, block-type walls, and oblique walls. Direct a ball towards exiting the grid by destroying oblique walls at precise moments in time.

The problem

You are playing a game on a $h \times w$ grid consisting of free cells, block-type walls, and oblique walls. Direct a ball towards exiting the grid by destroying oblique walls at precise moments in time.

Solution

- Analyze the structure of the problem!

I Pinball

AUTHORED BY: Lucian Bicsi PREPARED BY: Lucian Bicsi

The problem

You are playing a game on a $h \times w$ grid consisting of free cells, block-type walls, and oblique walls. Direct a ball towards exiting the grid by destroying oblique walls at precise moments in time.

Solution

- Analyze the structure of the problem!
- The movement of the ball can be grouped into (linear or circular) *regions*

I Pinball

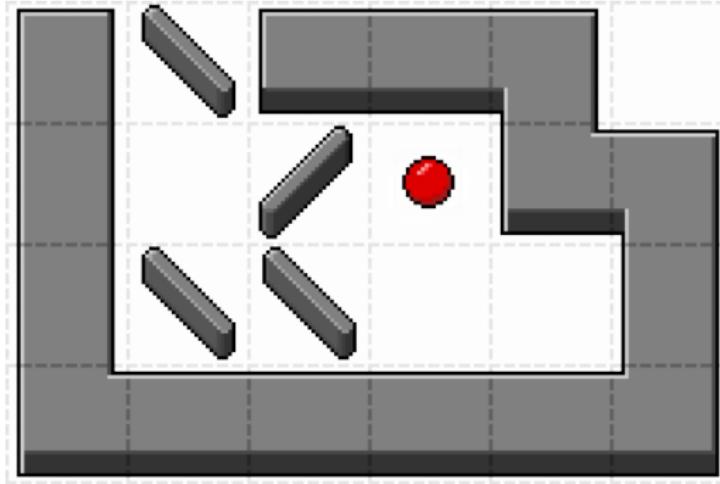
AUTHORED BY: Lucian Bicsi PREPARED BY: Lucian Bicsi

The problem

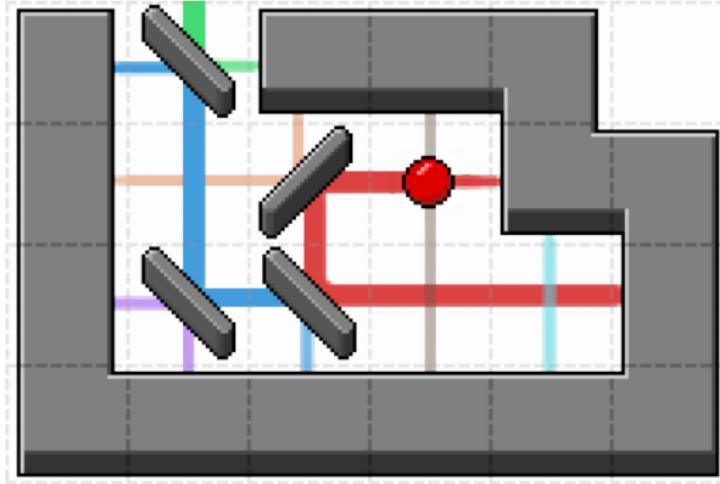
You are playing a game on a $h \times w$ grid consisting of free cells, block-type walls, and oblique walls. Direct a ball towards exiting the grid by destroying oblique walls at precise moments in time.

Solution

- Analyze the structure of the problem!
- The movement of the ball can be grouped into (linear or circular) *regions*
- Destroying a walls allows the ball to transition from a region to another region

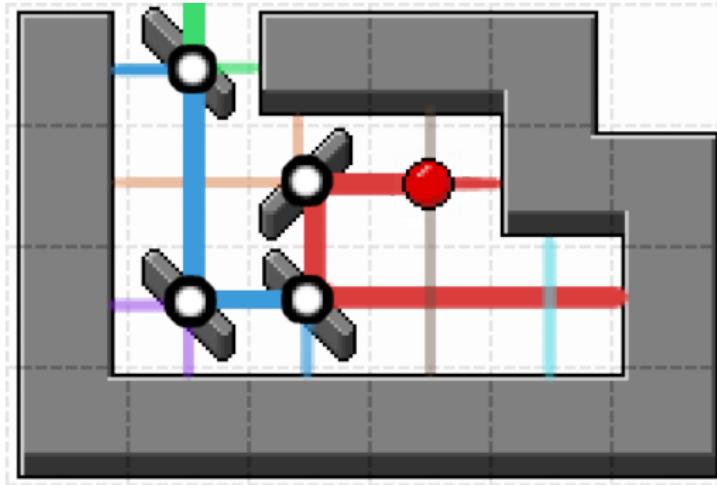


Solution



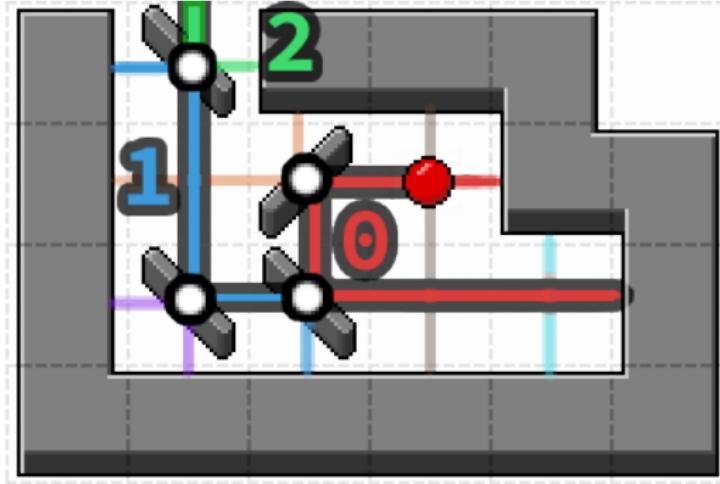
Solution

- Imagine a graph where each vertex is a region (red, blue, brown, cyan, purple, green)



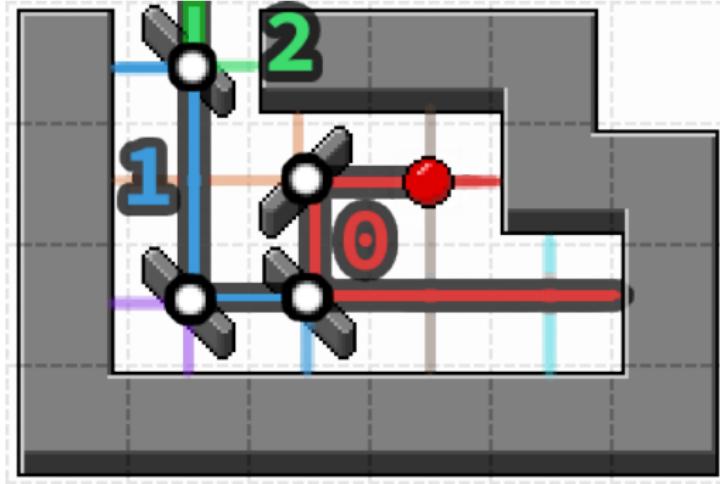
Solution

- Imagine a graph where each vertex is a region (red, blue, brown, cyan, purple, green)
 - Add an edge between two regions if they share an oblique wall (red-blue, red-orange, blue-purple, blue-green)



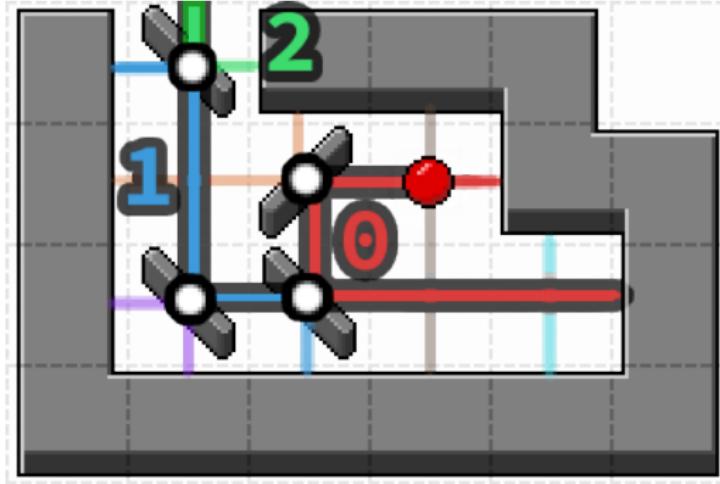
Solution

- Compute the minimum distance between any of the two starting (red, brown) regions and any of the regions that would allow the ball to escape (green)



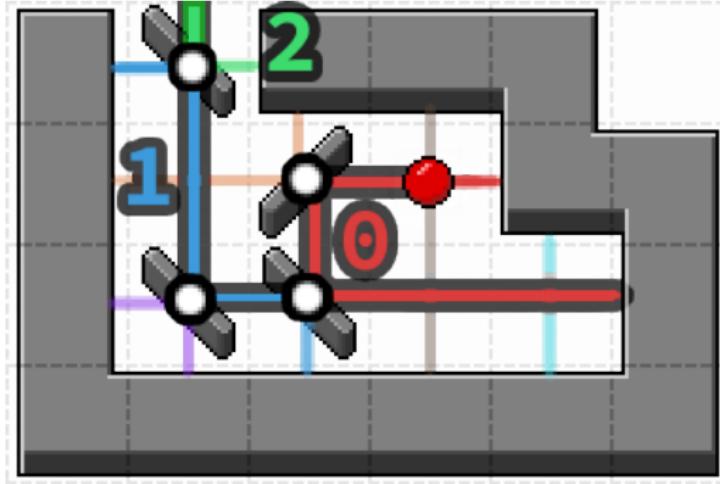
Solution

- Compute the minimum distance between any of the two starting (red, brown) regions and any of the regions that would allow the ball to escape (green)
- **Key fact!** The minimum distance is the answer to our problem!



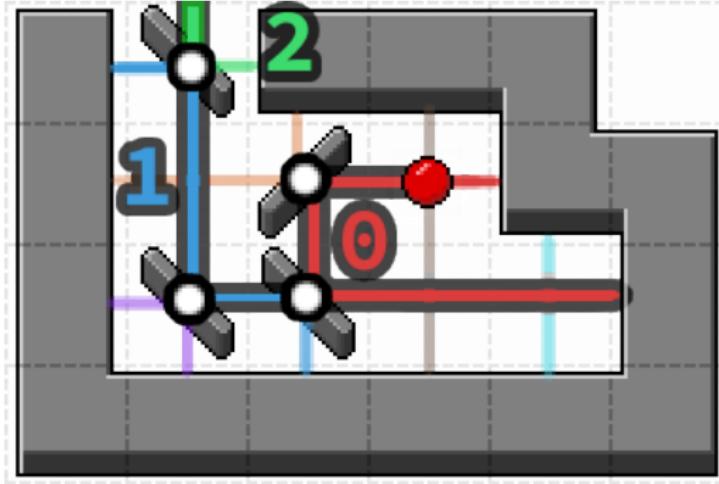
Proof

- The distance described is a *lower bound* for the solution.



Proof

- The distance described is a *lower bound* for the solution.
- We can prove, in fact, that *any cell* in a region at distance k can be reached with just k walls destroyed (not very trivial!)...



Proof

- The distance described is a *lower bound* for the solution.
- We can prove, in fact, that *any cell* in a region at distance k can be reached with just k walls destroyed (not very trivial!)...
- ... or we can just prove constructively!

Construction

- Multiple ways of handling reconstruction; some are easy, some are hard

Construction

- Multiple ways of handling reconstruction; some are easy, some are hard
- Most elegant: simulate in reverse order!

Construction

- Multiple ways of handling reconstruction; some are easy, some are hard
- Most elegant: simulate in reverse order!
 - “Destroying” a wall → “restoring” a wall

Construction

- Multiple ways of handling reconstruction; some are easy, some are hard
- Most elegant: simulate in reverse order!
 - “Destroying” a wall → “restoring” a wall
- Once the ball hits a wall:

Construction

- Multiple ways of handling reconstruction; some are easy, some are hard
- Most elegant: simulate in reverse order!
 - “Destroying” a wall → “restoring” a wall
- Once the ball hits a wall:
 - decide that it was, in fact, a destroyed wall and “restore” it;

Construction

- Multiple ways of handling reconstruction; some are easy, some are hard
- Most elegant: simulate in reverse order!
 - “Destroying” a wall → “restoring” a wall
- Once the ball hits a wall:
 - decide that it was, in fact, a destroyed wall and “restore” it;
 - or just continue simulating

Construction

- Multiple ways of handling reconstruction; some are easy, some are hard
- Most elegant: simulate in reverse order!
 - “Destroying” a wall → “restoring” a wall
- Once the ball hits a wall:
 - decide that it was, in fact, a destroyed wall and “restore” it;
 - or just continue simulating
- Simulate, simulate, simulate until the ball arrives back at its starting point

Construction

- Multiple ways of handling reconstruction; some are easy, some are hard
- Most elegant: simulate in reverse order!
 - “Destroying” a wall → “restoring” a wall
- Once the ball hits a wall:
 - decide that it was, in fact, a destroyed wall and “restore” it;
 - or just continue simulating
- Simulate, simulate, simulate until the ball arrives back at its starting point
- **Achtung!** Careful implementation helps solving this problem significantly