NWERC 2022

Solutions presentation

November 27, 2022

The NWERC 2022 Jury

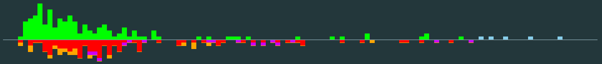
- **Bjarki Ágúst Guðmundsson**
  Google
- **Jorke de Vlas**
  Utrecht University
- **Ludo Pulles**
  Centrum Wiskunde & Informatica
  Amsterdam
- **Maarten Sijm**
  CHipCie (Delft University of Technology)
- **Markus Himmel**
  CAS Software, Karlsruhe
- **Michael Zündorf**
  Karlsruhe Institute of Technology
- **Nils Gustafsson**
  KTH Royal Institute of Technology
- **Paul Wild**
  FAU Erlangen-Nürnberg
- **Peter Kluit**
  Delft University of Technology
- **Ragnar Groot Koerkamp**
  ETH Zurich
- **Reinier Schmiermann**
  Utrecht University
- **Timon Knigge**
  ETH Zurich
- **Wendy Yi**
  Karlsruhe Institute of Technology

Big thanks to our test solvers

- **Bernhard Linn Hilmarsson**
  ETH Zurich
- **Bergur Snorrason**
  University of Iceland
- **Federico Glaudo**
  ETH Zurich
- **Henri Devillez**
  Université Catholique de Louvain
- **Joey Haas**
  Sioux Technologies

I: Interview Question

Problem Author: Paul Wild



Problem

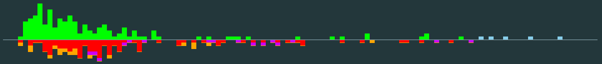
A group of players takes turns counting through the integers from c to d, except that

- each multiple of a is replaced by `Fizz`
- each multiple of b is replaced by `Buzz`
- each multiple of both a and b is replaced by `FizzBuzz`

Given a transcript of the game, reverse engineer the parameters a and b.

I: Interview Question

Problem Author: Paul Wild



Problem

A group of players takes turns counting through the integers from c to d, except that

- each multiple of a is replaced by `Fizz`
- each multiple of b is replaced by `Buzz`
- each multiple of both a and b is replaced by `FizzBuzz`

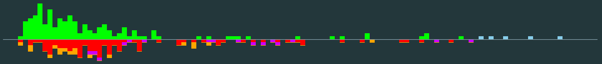
Given a transcript of the game, reverse engineer the parameters a and b.

Solution

- Find all the positions with `Fizz` (or `FizzBuzz`) and all the positions with `Buzz` (or `FizzBuzz`), then solve independently.

I: Interview Question

Problem Author: Paul Wild



Problem

A group of players takes turns counting through the integers from c to d, except that

- each multiple of a is replaced by `Fizz`
- each multiple of b is replaced by `Buzz`
- each multiple of both a and b is replaced by `FizzBuzz`

Given a transcript of the game, reverse engineer the parameters a and b.

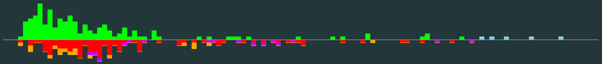
Solution

- Find all the positions with `Fizz` (or `FizzBuzz`) and all the positions with `Buzz` (or `FizzBuzz`), then solve independently.
- Three cases depending on the number of occurrences:

2 or more	$\leadsto$	output the difference between the first two occurrences.
1	$\leadsto$	output the position of that single occurrence.
0	$\leadsto$	output some number past the end of the range.

I: Interview Question

Problem Author: Paul Wild



Problem

A group of players takes turns counting through the integers from c to d, except that

- each multiple of a is replaced by `Fizz`
- each multiple of b is replaced by `Buzz`
- each multiple of both a and b is replaced by `FizzBuzz`

Given a transcript of the game, reverse engineer the parameters a and b.

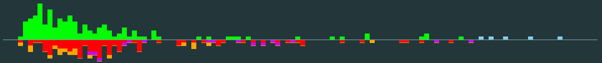
Pitfalls

Exceptions in Java are not fast enough...

```
try { int v = Integer.parseInt(s); } catch (NumberFormatException e) { ... }
```

I: Interview Question

Problem Author: Paul Wild



Problem

A group of players takes turns counting through the integers from c to d, except that

- each multiple of a is replaced by `Fizz`
- each multiple of b is replaced by `Buzz`
- each multiple of both a and b is replaced by `FizzBuzz`

Given a transcript of the game, reverse engineer the parameters a and b.

Pitfalls

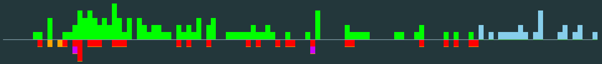
Exceptions in Java are not fast enough...

```
try { int v = Integer.parseInt(s); } catch (NumberFormatException e) { ... }
```

Statistics: 268 submissions, 136 accepted, 5 unknown

B: Bottle Flip

Problem Author: Jorke de Vlas



Problem

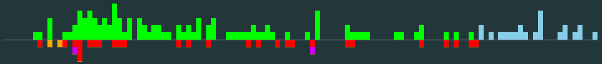
Given:

- the density d_a of air and d_w of water,
- the radius r and height h of a cylindrical container.

To which height should the cylinder be filled with water to minimise the height of the centre of mass?

B: Bottle Flip

Problem Author: Jorke de Vlas

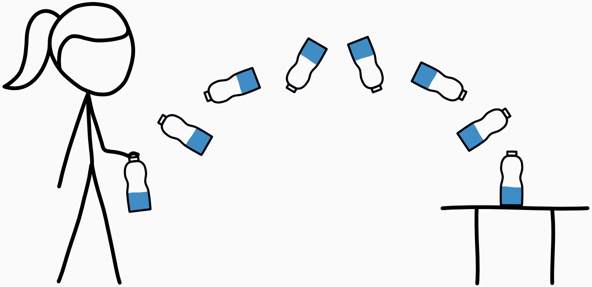


Problem

Given:

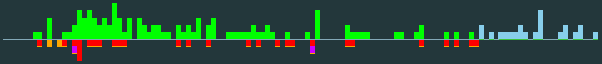
- the density d_a of air and d_w of water,
- the radius r and height h of a cylindrical container.

To which height should the cylinder be filled with water to minimise the height of the centre of mass?



B: Bottle Flip

Problem Author: Jorke de Vlas

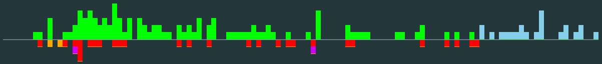


Observations

- The radius r is irrelevant.

B: Bottle Flip

Problem Author: Jorke de Vlas

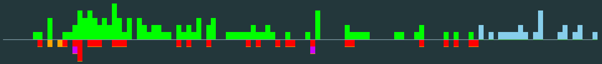


Observations

- The radius r is irrelevant.
- The result can be found using ternary search.

B: Bottle Flip

Problem Author: Jorke de Vlas



Observations

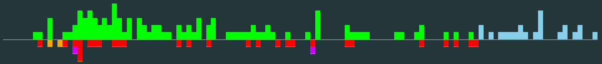
- The radius r is irrelevant.
- The result can be found using ternary search.

Solution

- Given the height h_w, calculate $h_a = h - h_w$.
- The centre of mass of the water is at height $c_w = \frac{h_w}{2}$.
- The centre of mass of the air is at height $c_a = h - \frac{h_a}{2}$.

B: Bottle Flip

Problem Author: Jorke de Vlas



Observations

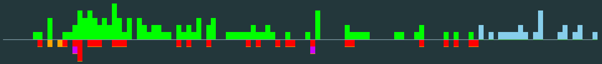
- The radius r is irrelevant.
- The result can be found using ternary search.

Solution

- Given the height h_w, calculate $h_a = h - h_w$.
- The centre of mass of the water is at height $c_w = \frac{h_w}{2}$.
- The centre of mass of the air is at height $c_a = h - \frac{h_a}{2}$.
- The height of the combined centre of mass is the weighted average

B: Bottle Flip

Problem Author: Jorke de Vlas



Observations

- The radius r is irrelevant.
- The result can be found using ternary search.

Solution

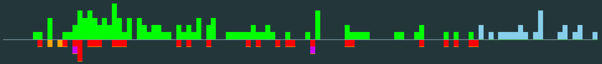
- Given the height h_w, calculate $h_a = h - h_w$.
- The centre of mass of the water is at height $c_w = \frac{h_w}{2}$.
- The centre of mass of the air is at height $c_a = h - \frac{h_a}{2}$.
- The height of the combined centre of mass is the weighted average:

$$\frac{c_a \cdot d_a \cdot h_a + c_w \cdot d_w \cdot h_w}{h_a \cdot d_a + h_w \cdot d_w}.$$

- Can also be found by differentiating a nasty expression (left as an exercise for the reader).

B: Bottle Flip

Problem Author: Jorke de Vlas



Observations

- The radius r is irrelevant.
- The result can be found using ternary search.

Solution

- Given the height h_w, calculate $h_a = h - h_w$.
- The centre of mass of the water is at height $c_w = \frac{h_w}{2}$.
- The centre of mass of the air is at height $c_a = h - \frac{h_a}{2}$.
- The height of the combined centre of mass is the weighted average:

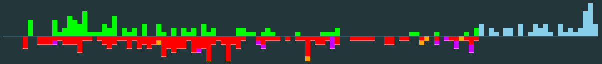
$$\frac{c_a \cdot d_a \cdot h_a + c_w \cdot d_w \cdot h_w}{h_a \cdot d_a + h_w \cdot d_w}.$$

- Can also be found by differentiating a nasty expression (left as an exercise for the reader).

Statistics: 154 submissions, 100 accepted, 22 unknown

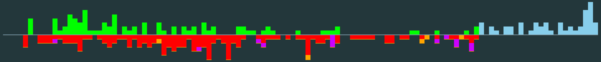
C: Circular Caramel Cookie

Problem Author: Maarten Sijm



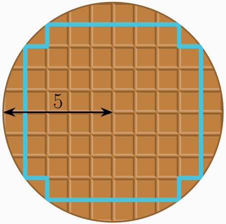
Problem

Given an integer s, output the minimum radius of a circle that contains $> s$ whole unit squares.



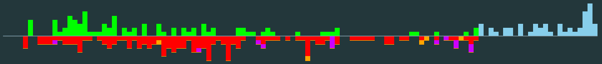
Problem

Given an integer s, output the minimum radius of a circle that contains $> s$ whole unit squares.



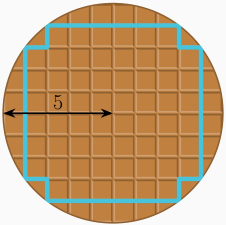
C: Circular Caramel Cookie

Problem Author: Maarten Sijm



Problem

Given an integer s, output the minimum radius of a circle that contains $> s$ whole unit squares.

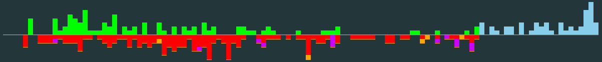


Solution

- For a fixed radius r, we can determine the number of whole unit squares that fit in the circle.

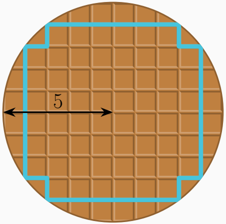
C: Circular Caramel Cookie

Problem Author: Maarten Sijm



Problem

Given an integer s, output the minimum radius of a circle that contains $> s$ whole unit squares.

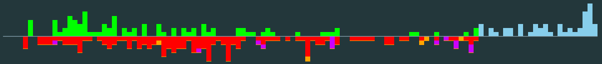


Solution

- For a fixed radius r, we can determine the number of whole unit squares that fit in the circle.
- Determine how many squares fit in each column using the Pythagorean Theorem. ($\mathcal{O}(\sqrt{s})$)

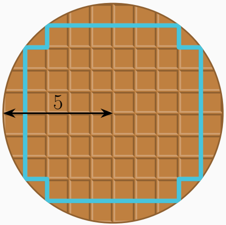
C: Circular Caramel Cookie

Problem Author: Maarten Sijm



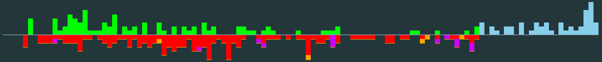
Problem

Given an integer s, output the minimum radius of a circle that contains $> s$ whole unit squares.



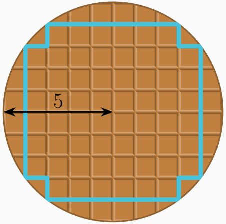
Solution

- For a fixed radius r, we can determine the number of whole unit squares that fit in the circle.
- Determine how many squares fit in each column using the Pythagorean Theorem. ($\mathcal{O}(\sqrt{s})$)
- Use binary search to find the solution. Total time: $\mathcal{O}(\log s \cdot \sqrt{s})$.



Problem

Given an integer s, output the minimum radius of a circle that contains $> s$ whole unit squares.

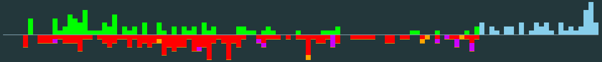


Challenge

It is possible in $\mathcal{O}(\sqrt{s})$ as well.

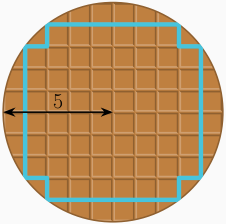
C: Circular Caramel Cookie

Problem Author: Maarten Sijm



Problem

Given an integer s, output the minimum radius of a circle that contains $> s$ whole unit squares.



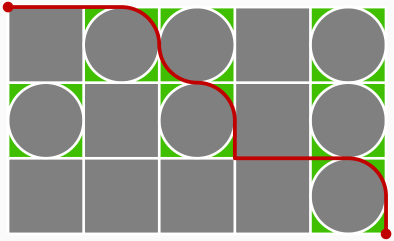
Challenge

It is possible in $\mathcal{O}(\sqrt{s})$ as well.

Statistics: 298 submissions, 89 accepted, 49 unknown

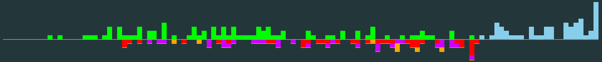
Problem

Find the shortest path from the north-west to the south-east on a map of Delft with round towers and square buildings.



D: Delft Distance

Problem Author: Reinier Schmiermann

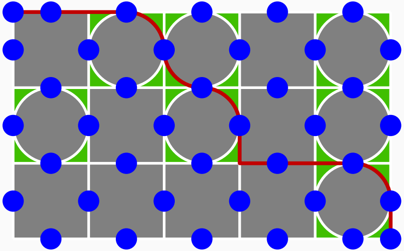


Problem

Find the shortest path from the north-west to the south-east on a map of Delft with round towers and square buildings.

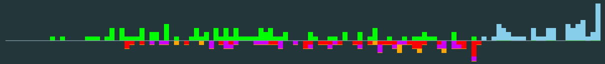
Observation

Not all points on the map need to be checked:



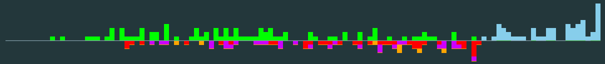
D: Delft Distance

Problem Author: Reinier Schmiermann



Solution 1: Dijkstra

- Turn the map into a graph,
  - straight edges are 10 m, and
  - round edges are 5π m.

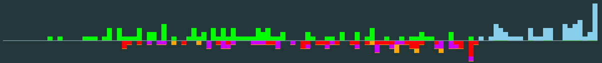


Solution 1: Dijkstra

- Turn the map into a graph,
  - straight edges are 10 m, and
  - round edges are 5π m.
- Running Dijkstra takes $\mathcal{O}(n \log n)$ time $(n = w \cdot h)$.

D: Delft Distance

Problem Author: Reinier Schmiermann



Solution 1: Dijkstra

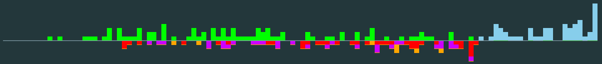
- Turn the map into a graph,
  - straight edges are 10 m, and
  - round edges are 5π m.
- Running Dijkstra takes $\mathcal{O}(n \log n)$ time ($n = w \cdot h$).

Solution 2: Dynamic Programming

- For every blue vertex (left-to-right, then top-to-bottom), take the minimum between
  - going straight across (right or down) and
  - going across a corner (right-and-down or down-and-right).

D: Delft Distance

Problem Author: Reinier Schmiermann



Solution 1: Dijkstra

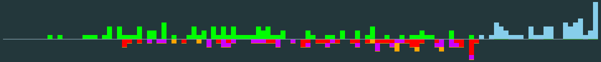
- Turn the map into a graph,
  - straight edges are 10 m, and
  - round edges are 5π m.
- Running Dijkstra takes $\mathcal{O}(n \log n)$ time ($n = w \cdot h$).

Solution 2: Dynamic Programming

- For every blue vertex (left-to-right, then top-to-bottom), take the minimum between
  - going straight across (right or down) and
  - going across a corner (right-and-down or down-and-right).
- This takes $\mathcal{O}(n)$ time ($n = w \cdot h$).

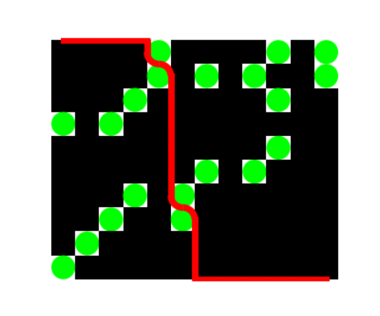
D: Delft Distance

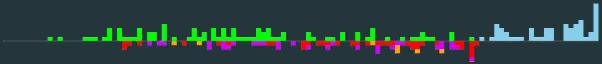
Problem Author: Reinier Schmiermann



Problem

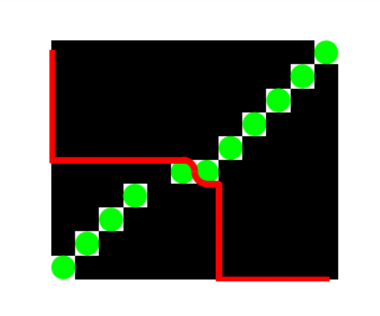
Find the shortest path from the north-west to the south-east on a map of Delft with round towers and square buildings.





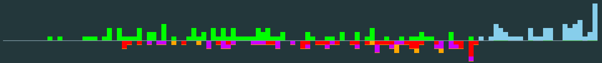
Problem

Find the shortest path from the north-west to the south-east on a map of Delft with round towers and square buildings.



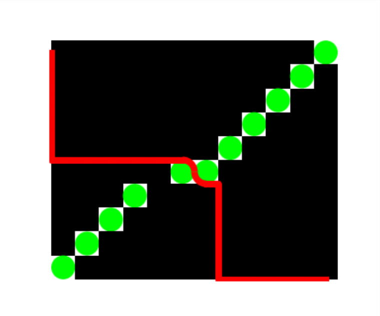
D: Delft Distance

Problem Author: Reinier Schmiermann



Problem

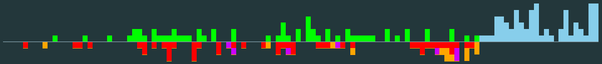
Find the shortest path from the north-west to the south-east on a map of Delft with round towers and square buildings.



Statistics: 215 submissions, 85 accepted, 53 unknown

E: ETA

Problem Author: Paul Wild

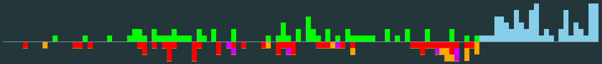


Problem

Construct a graph such that the average optimal time to reach vertex 1 is exactly $\frac{a}{b}$ or determine that this is impossible.

E: ETA

Problem Author: Paul Wild



Problem

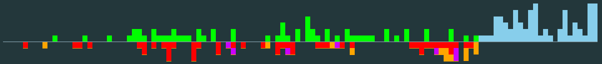
Construct a graph such that the average optimal time to reach vertex 1 is exactly $\frac{a}{b}$ or determine that this is impossible.

Observations

- Only edges on an optimal path to vertex 1 are relevant, so without loss of generality the graph is a tree.

E: ETA

Problem Author: Paul Wild



Problem

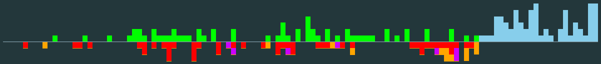
Construct a graph such that the average optimal time to reach vertex 1 is exactly $\frac{a}{b}$ or determine that this is impossible.

Observations

- Only edges on an optimal path to vertex 1 are relevant, so without loss of generality the graph is a tree.
- The exact shape of this tree does not matter, only the number of vertices in each layer.

E: ETA

Problem Author: Paul Wild



Problem

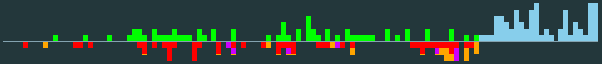
Construct a graph such that the average optimal time to reach vertex 1 is exactly $\frac{a}{b}$ or determine that this is impossible.

Observations

- Only edges on an optimal path to vertex 1 are relevant, so without loss of generality the graph is a tree.
- The exact shape of this tree does not matter, only the number of vertices in each layer.
- Represent the graph as a list $(a_0, a_1, \ldots, a_k)$ where a_i is the number of vertices in layer i

E: ETA

Problem Author: Paul Wild



Problem

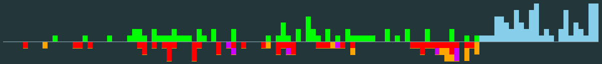
Construct a graph such that the average optimal time to reach vertex 1 is exactly $\frac{a}{b}$ or determine that this is impossible.

Observations

- Only edges on an optimal path to vertex 1 are relevant, so without loss of generality the graph is a tree.
- The exact shape of this tree does not matter, only the number of vertices in each layer.
- Represent the graph as a list $(a_0, a_1, \ldots, a_k)$ where a_i is the number of vertices in layer i

E: ETA

Problem Author: Paul Wild



Problem

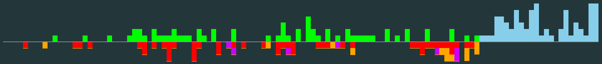
Construct a graph such that the average optimal time to reach vertex 1 is exactly $\frac{a}{b}$ or determine that this is impossible.

Observations

- Only edges on an optimal path to vertex 1 are relevant, so without loss of generality the graph is a tree.
- The exact shape of this tree does not matter, only the number of vertices in each layer.
- Represent the graph as a list $(a_0, a_1, \ldots, a_k)$ where a_i is the number of vertices in layer i, satisfying:
  - There is only 1 vertex at the root layer, so $a_0 = 1$.
  - There can only be vertices at layer x if there are some at layer $x - 1$, so for every i, $a_i \geq 1$.

E: ETA

Problem Author: Paul Wild



Problem

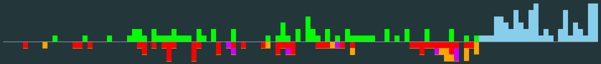
Construct a graph such that the average optimal time to reach vertex 1 is exactly $\frac{a}{b}$ or determine that this is impossible.

Solution

- The graph can be represented as a list $(a_0, a_1, \ldots, a_k)$ where a_i is the number of vertices in layer i.

E: ETA

Problem Author: Paul Wild



Problem

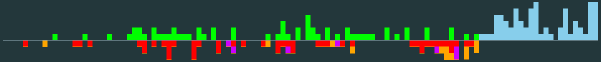
Construct a graph such that the average optimal time to reach vertex 1 is exactly $\frac{a}{b}$ or determine that this is impossible.

Solution

- The graph can be represented as a list $(a_0, a_1, \ldots, a_k)$ where a_i is the number of vertices in layer i.
- Given such a list, construct a graph: vertex 1 is the root, and vertices at layer i have a single vertex at layer $i - 1$ as parent.

E: ETA

Problem Author: Paul Wild

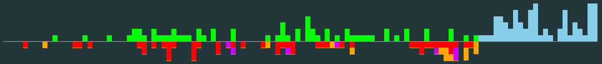


Problem

Construct a graph such that the average optimal time to reach vertex 1 is exactly $\frac{a}{b}$ or determine that this is impossible.

Solution

- The graph can be represented as a list $(a_0, a_1, \ldots, a_k)$ where a_i is the number of vertices in layer i.
- Given such a list, construct a graph: vertex 1 is the root, and vertices at layer i have a single vertex at layer $i-1$ as parent.
- The total number of vertices is $a_0 + a_1 + \ldots + a_k$.



Problem

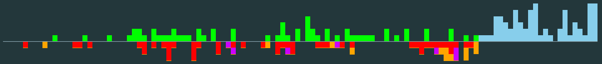
Construct a graph such that the average optimal time to reach vertex 1 is exactly $\frac{a}{b}$ or determine that this is impossible.

Solution

- The graph can be represented as a list $(a_0, a_1, \ldots, a_k)$ where a_i is the number of vertices in layer i.
- Given such a list, construct a graph: vertex 1 is the root, and vertices at layer i have a single vertex at layer $i - 1$ as parent.
- The total number of vertices is $a_0 + a_1 + \ldots + a_k$.
- The optimal time for a vertex at layer i is i, so the average optimal time is $\frac{0 \cdot a_0 + 1 \cdot a_1 + \ldots + k \cdot a_k}{a_0 + a_1 + \ldots + a_k}$.

E: ETA

Problem Author: Paul Wild



Problem

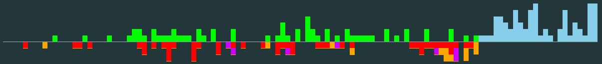
Construct a graph such that the average optimal time to reach vertex 1 is exactly $\frac{a}{b}$ or determine that this is impossible.

Solution

- The graph can be represented as a list $(a_0, a_1, \ldots, a_k)$ where a_i is the number of vertices in layer i.
- Given such a list, construct a graph: vertex 1 is the root, and vertices at layer i have a single vertex at layer $i - 1$ as parent.
- The total number of vertices is $a_0 + a_1 + \ldots + a_k$.
- The optimal time for a vertex at layer i is i, so the average optimal time is $\frac{0 \cdot a_0 + 1 \cdot a_1 + \ldots + k \cdot a_k}{a_0 + a_1 + \ldots + a_k}$.
- We consider two cases: either $\frac{a}{b} < 1$ or $\frac{a}{b} \geq 1$.

E: ETA

Problem Author: Paul Wild



Problem

Construct a graph such that the average optimal time to reach vertex 1 is exactly $\frac{a}{b}$ or determine that this is impossible.

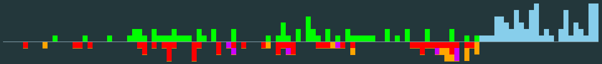
Solution

Case 1: $\frac{a}{b} < 1$.

- If there is a vertex with optimal time at least 2, then the average optimal time is at least 1. Thus, such vertices cannot exist.

E: ETA

Problem Author: Paul Wild



Problem

Construct a graph such that the average optimal time to reach vertex 1 is exactly $\frac{a}{b}$ or determine that this is impossible.

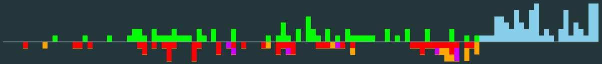
Solution

Case 1: $\frac{a}{b} < 1$.

- If there is a vertex with optimal time at least 2, then the average optimal time is at least 1. Thus, such vertices cannot exist.
- The average optimal time is now $\frac{a_1}{1+a_1}$.

E: ETA

Problem Author: Paul Wild



Problem

Construct a graph such that the average optimal time to reach vertex 1 is exactly $\frac{a}{b}$ or determine that this is impossible.

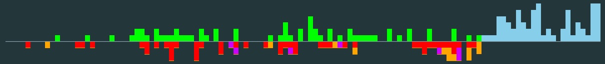
Solution

Case 1: $\frac{a}{b} < 1$.

- If there is a vertex with optimal time at least 2, then the average optimal time is at least 1. Thus, such vertices cannot exist.
- The average optimal time is now $\frac{a_1}{1+a_1}$.
- If $a = b - 1$, we solve the problem with the list $(1, a)$. Otherwise, the answer is `impossible`.

E: ETA

Problem Author: Paul Wild



Problem

Construct a graph such that the average optimal time to reach vertex 1 is exactly $\frac{a}{b}$ or determine that this is impossible.

Solution

Case 2: $\frac{a}{b} \geq 1$. Define k as $\lfloor \frac{a}{b} \rfloor$.

- Consider a list of length $2k + 1$ where every a_i is 1 except for a_k. We set a_k to a value such that $a_k > 2k + 1$ and the total number of vertices is divisible by b, i.e. $n = m \cdot b$.

Problem

Construct a graph such that the average optimal time to reach vertex 1 is exactly $\frac{a}{b}$ or determine that this is impossible.

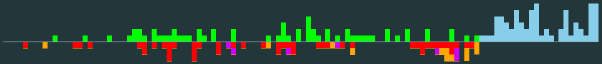
Solution

Case 2: $\frac{a}{b} \geq 1$. Define k as $\lfloor \frac{a}{b} \rfloor$.

- Consider a list of length $2k+1$ where every a_i is 1 except for a_k. We set a_k to a value such that $a_k > 2k+1$ and the total number of vertices is divisible by b, i.e. $n = m \cdot b$.
- The average optimal time is $k \leq \frac{a}{b}$: all the ones cancel each other out.

E: ETA

Problem Author: Paul Wild



Problem

Construct a graph such that the average optimal time to reach vertex 1 is exactly $\frac{a}{b}$ or determine that this is impossible.

Solution

Case 2: $\frac{a}{b} \geq 1$. Define k as $\lfloor \frac{a}{b} \rfloor$.

- Consider a list of length $2k+1$ where every a_i is 1 except for a_k. We set a_k to a value such that $a_k > 2k+1$ and the total number of vertices is divisible by b, i.e. $n = m \cdot b$.
- The average optimal time is $k \leq \frac{a}{b}$: all the ones cancel each other out.
- Moving a vertex one layer up increases the average by $\frac{1}{nb}$. Moving $(\frac{a}{b} - k) \cdot nb$ vertices increases it to $\frac{a}{b}$.

Problem

Construct a graph such that the average optimal time to reach vertex 1 is exactly $\frac{a}{b}$ or determine that this is impossible.

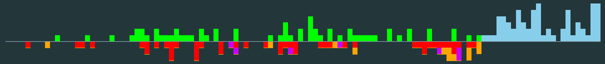
Solution

Case 2: $\frac{a}{b} \geq 1$. Define k as $\lfloor \frac{a}{b} \rfloor$.

- Consider a list of length $2k+1$ where every a_i is 1 except for a_k. We set a_k to a value such that $a_k > 2k+1$ and the total number of vertices is divisible by b, i.e. $n = m \cdot b$.
- The average optimal time is $k \leq \frac{a}{b}$: all the ones cancel each other out.
- Moving a vertex one layer up increases the average by $\frac{1}{nb}$. Moving $(\frac{a}{b} - k) \cdot nb$ vertices increases it to $\frac{a}{b}$.
- Such movements are possible: over half of the vertices is at layer k, so moving those to layer $k+2$ increases the average by 1, which is already too much.

E: ETA

Problem Author: Paul Wild



Problem

Construct a graph such that the average optimal time to reach vertex 1 is exactly $\frac{a}{b}$ or determine that this is impossible.

Solution

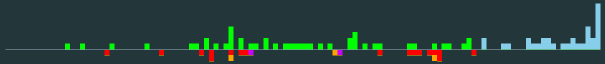
Case 2: $\frac{a}{b} \geq 1$. Define k as $\lfloor \frac{a}{b} \rfloor$.

- Consider a list of length $2k+1$ where every a_i is 1 except for a_k. We set a_k to a value such that $a_k > 2k+1$ and the total number of vertices is divisible by b, i.e. $n = m \cdot b$.
- The average optimal time is $k \leq \frac{a}{b}$: all the ones cancel each other out.
- Moving a vertex one layer up increases the average by $\frac{1}{nb}$. Moving $(\frac{a}{b} - k) \cdot nb$ vertices increases it to $\frac{a}{b}$.
- Such movements are possible: over half of the vertices is at layer k, so moving those to layer $k+2$ increases the average by 1, which is already too much.

Statistics: 196 submissions, 62 accepted, 67 unknown

H: High-quality Tree

Problem Author: Michael Zündorf

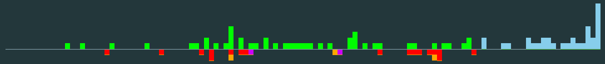


Problem

Given a binary tree, determine the minimal number of leaves you should remove to make the tree strongly balanced.

H: High-quality Tree

Problem Author: Michael Zündorf



Problem

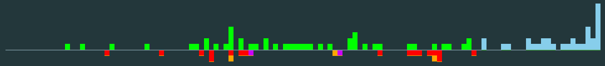
Given a binary tree, determine the minimal number of leaves you should remove to make the tree strongly balanced.

Solution

- Every vertex should be balanced: the height of its left and right subtree should differ by at most one.

H: High-quality Tree

Problem Author: Michael Zündorf



Problem

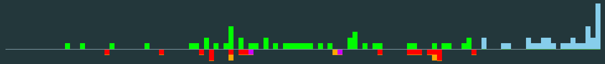
Given a binary tree, determine the minimal number of leaves you should remove to make the tree strongly balanced.

Solution

- Every vertex should be balanced: the height of its left and right subtree should differ by at most one.
- Naive solution: remove the deepest leaves below vertices that are too high.

H: High-quality Tree

Problem Author: Michael Zündorf



Problem

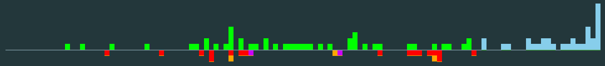
Given a binary tree, determine the minimal number of leaves you should remove to make the tree strongly balanced.

Solution

- Every vertex should be balanced: the height of its left and right subtree should differ by at most one.
- Naive solution: remove the deepest leaves below vertices that are too high.
- This takes $\mathcal{O}(n)$ time per vertex, so too slow.

H: High-quality Tree

Problem Author: Michael Zündorf



Problem

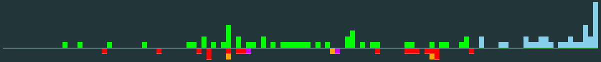
Given a binary tree, determine the minimal number of leaves you should remove to make the tree strongly balanced.

Solution

- Idea: determine the maximal height every subtree can have, and then remove vertices.

H: High-quality Tree

Problem Author: Michael Zündorf



Problem

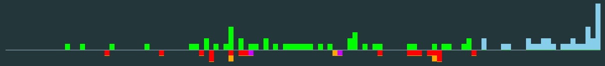
Given a binary tree, determine the minimal number of leaves you should remove to make the tree strongly balanced.

Solution

- Idea: determine the maximal height every subtree can have, and then remove vertices.
- First, compute all heights using a DFS.

H: High-quality Tree

Problem Author: Michael Zündorf



Problem

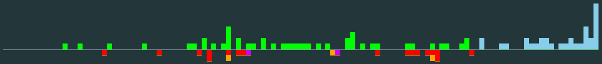
Given a binary tree, determine the minimal number of leaves you should remove to make the tree strongly balanced.

Solution

- Idea: determine the maximal height every subtree can have, and then remove vertices.
- First, compute all heights using a DFS.
- Set the required heights using a second DFS. For a vertex v with children l and r, the minimal required height of l is: $\min(H(l), H(r)+1, ReqH(v)-1)$. Analogous for r.

H: High-quality Tree

Problem Author: Michael Zündorf



Problem

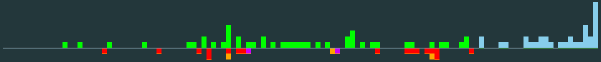
Given a binary tree, determine the minimal number of leaves you should remove to make the tree strongly balanced.

Solution

- Idea: determine the maximal height every subtree can have, and then remove vertices.
- First, compute all heights using a DFS.
- Set the required heights using a second DFS. For a vertex v with children l and r, the minimal required height of l is: $\min(H(l), H(r) + 1, ReqH(v) - 1)$. Analogous for r.
- Finally, remove all vertices with negative height.

H: High-quality Tree

Problem Author: Michael Zündorf



Problem

Given a binary tree, determine the minimal number of leaves you should remove to make the tree strongly balanced.

Solution

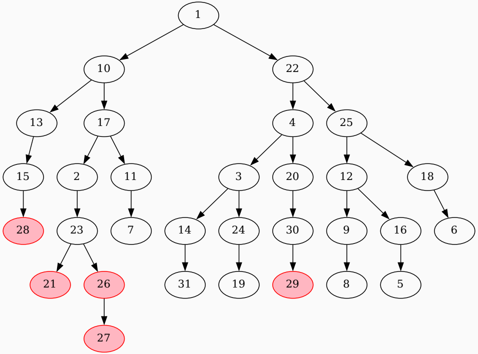
- Idea: determine the maximal height every subtree can have, and then remove vertices.
- First, compute all heights using a DFS.
- Set the required heights using a second DFS. For a vertex v with children l and r, the minimal required height of l is: $\min(H(l), H(r)+1, ReqH(v)-1)$. Analogous for r.
- Finally, remove all vertices with negative height.
- Runtime: $\mathcal{O}(n)$

H: High-quality Tree

Problem Author: Michael Zündorf

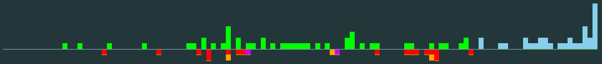
Problem

Given a binary tree, determine the minimal number of leaves you should remove to make the tree strongly balanced.



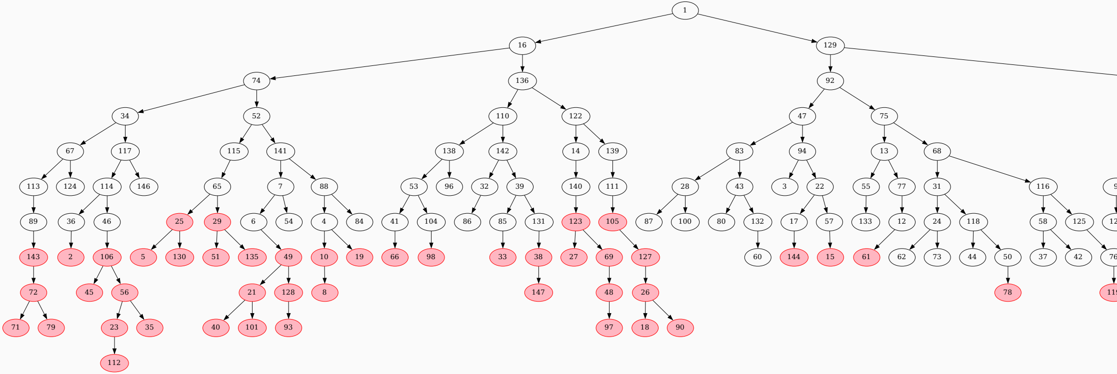
H: High-quality Tree

Problem Author: Michael Zündorf



Problem

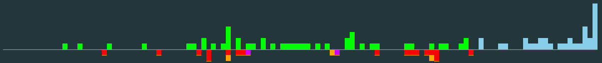
Given a binary tree, determine the minimal number of leaves you should remove to make the tree strongly balanced.



Statistics: 100 submissions, 45 accepted, 33 unknown

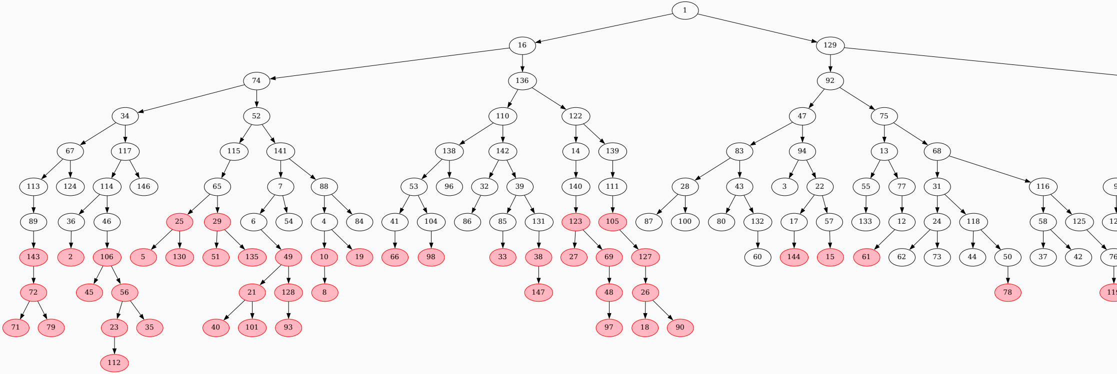
H: High-quality Tree

Problem Author: Michael Zündorf



Problem

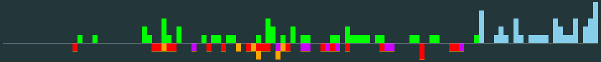
Given a binary tree, determine the minimal number of leaves you should remove to make the tree strongly balanced.



Statistics: 100 submissions, 45 accepted, 33 unknown

J: Justice Served

Problem Author: Michael Zündorf

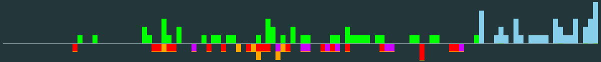


Problem

Given n intervals $I_i = [\ell_i, r_i]$, for each of them find the length $v(I_i)$ of the longest *chain* $I_i \subset I_{i_1} \subset I_{i_2} \subset \dots$.

J: Justice Served

Problem Author: Michael Zündorf



Problem

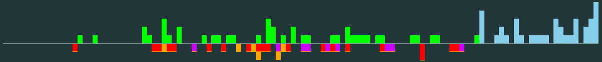
Given n intervals $I_i = [\ell_i, r_i]$, for each of them find the length $v(I_i)$ of the longest *chain* $I_i \subset I_{i_1} \subset I_{i_2} \subset \dots$.

Naive solution

- $I_i \subset I_j$ is only possible if $r_i - \ell_i = t_i < t_j = r_j - \ell_j$.

J: Justice Served

Problem Author: Michael Zündorf

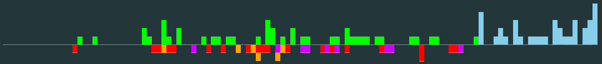


Problem

Given n intervals $I_i = [\ell_i, r_i]$, for each of them find the length $v(I_i)$ of the longest *chain* $I_i \subset I_{i_1} \subset I_{i_2} \subset \ldots$.

Naive solution

- $I_i \subset I_j$ is only possible if $r_i - \ell_i = t_i < t_j = r_j - \ell_j$.
- Sort by decreasing length and iterate over all longer intervals $\to \mathcal{O}(n^2)$.



Problem

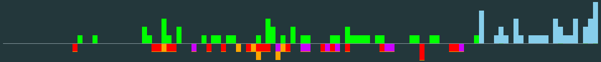
Given n intervals $I_i = [\ell_i, r_i]$, for each of them find the length $v(I_i)$ of the longest *chain* $I_i \subset I_{i_1} \subset I_{i_2} \subset \dots$.

Solution

- Sort by increasing ℓ first, and then decreasing r.

J: Justice Served

Problem Author: Michael Zündorf



Problem

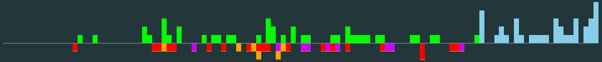
Given n intervals $I_i = [\ell_i, r_i]$, for each of them find the length $v(I_i)$ of the longest *chain* $I_i \subset I_{i_1} \subset I_{i_2} \subset \dots$.

Solution

- Sort by increasing ℓ first, and then decreasing r.
- The value $v(I_i)$ of $[\ell_i, r_i]$ is $1 + \max_{\ell_j \leq \ell_i, r_i \leq r_j} v(r_j)$.

J: Justice Served

Problem Author: Michael Zündorf



Problem

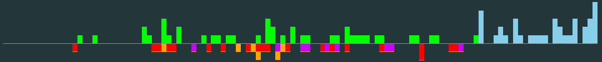
Given n intervals $I_i = [\ell_i, r_i]$, for each of them find the length $v(I_i)$ of the longest *chain* $I_i \subset I_{i_1} \subset I_{i_2} \subset \dots$.

Solution

- Sort by increasing ℓ first, and then decreasing r.
- The value $v(I_i)$ of $[\ell_i, r_i]$ is $1 + \max_{\ell_j \leq \ell_i, r_i \leq r_j} v(r_j)$.
- Ignore r_i if $r_i < r_j$ and $v(I_i) \leq v(I_j)$.

J: Justice Served

Problem Author: Michael Zündorf



Problem

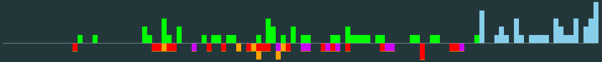
Given n intervals $I_i = [\ell_i, r_i]$, for each of them find the length $v(I_i)$ of the longest *chain* $I_i \subset I_{i_1} \subset I_{i_2} \subset \dots$.

Solution

- Sort by increasing ℓ first, and then decreasing r.
- The value $v(I_i)$ of $[\ell_i, r_i]$ is $1 + \max_{\ell_j \le \ell_i, r_i \le r_j} v(r_j)$.
- Ignore r_i if $r_i < r_j$ and $v(I_i) \le v(I_j)$.
- What is left are increasing r_i with decreasing v, that can be stored in an ordered set.

J: Justice Served

Problem Author: Michael Zündorf



Problem

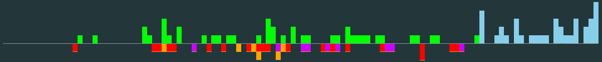
Given n intervals $I_i = [\ell_i, r_i]$, for each of them find the length $v(I_i)$ of the longest *chain* $I_i \subset I_{i_1} \subset I_{i_2} \subset \dots$.

Solution

- Sort by increasing ℓ first, and then decreasing r.
- The value $v(I_i)$ of $[\ell_i, r_i]$ is $1 + \max_{\ell_j \le \ell_i, r_i \le r_j} v(r_j)$.
- Ignore r_i if $r_i < r_j$ and $v(I_i) \le v(I_j)$.
- What is left are increasing r_i with decreasing v, that can be stored in an ordered set.
- Compute $v(I_i)$ by looking up the first element at least r_i.

J: Justice Served

Problem Author: Michael Zündorf



Problem

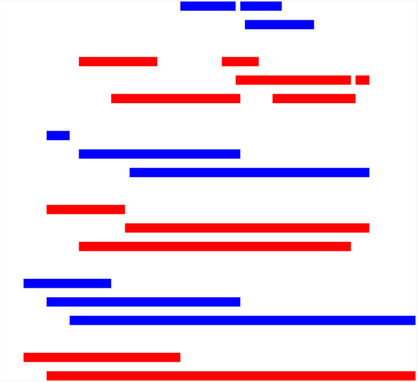
Given n intervals $I_i = [\ell_i, r_i]$, for each of them find the length $v(I_i)$ of the longest *chain* $I_i \subset I_{i_1} \subset I_{i_2} \subset \dots$.

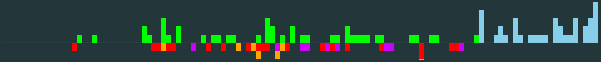
Solution

- Sort by increasing ℓ first, and then decreasing r.
- The value $v(I_i)$ of $[\ell_i, r_i]$ is $1 + \max_{\ell_j \le \ell_i, r_i \le r_j} v(r_j)$.
- Ignore r_i if $r_i < r_j$ and $v(I_i) \le v(I_j)$.
- What is left are increasing r_i with decreasing v, that can be stored in an ordered set.
- Compute $v(I_i)$ by looking up the first element at least r_i.
- Insert $v(I_i)$ into the set and remove new suboptimal points that follow it.

Problem

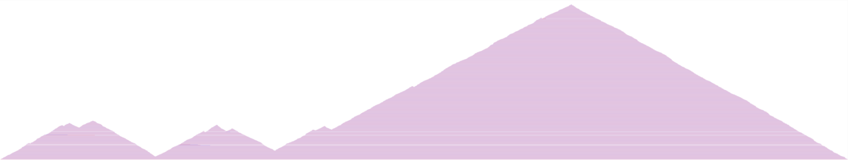
Given n intervals $I_i = [\ell_i, r_i]$, for each of them find the length $v(I_i)$ of the longest *chain* $I_i \subset I_{i_1} \subset I_{i_2} \subset \dots$.

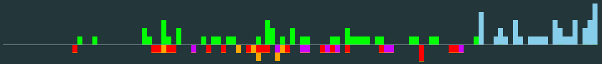




Problem

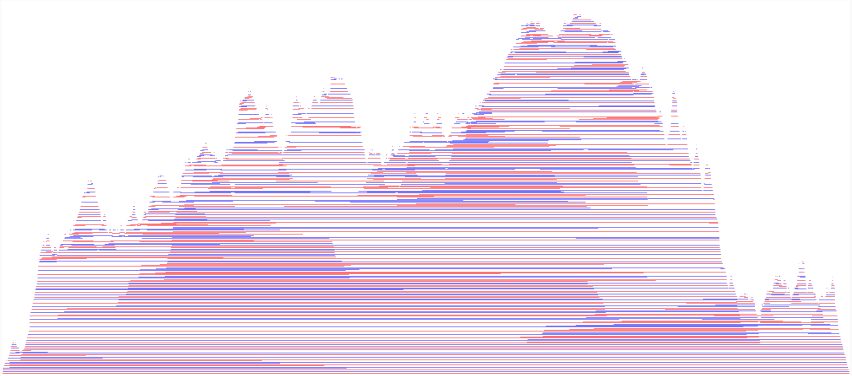
Given n intervals $I_i = [\ell_i, r_i]$, for each of them find the length $v(I_i)$ of the longest *chain* $I_i \subset I_{i_1} \subset I_{i_2} \subset \dots$.





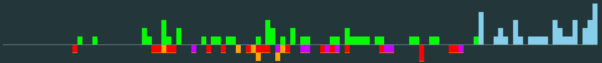
Problem

Given n intervals $I_i = [\ell_i, r_i]$, for each of them find the length $v(I_i)$ of the longest *chain* $I_i \subset I_{i_1} \subset I_{i_2} \subset \dots$.



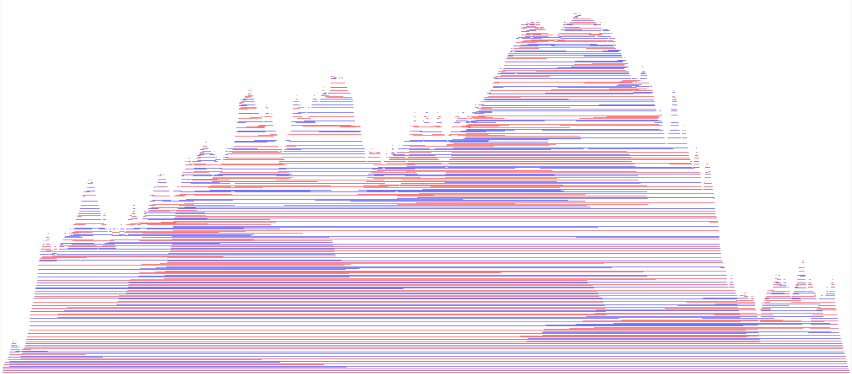
J: Justice Served

Problem Author: Michael Zündorf



Problem

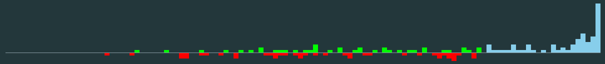
Given n intervals $I_i = [\ell_i, r_i]$, for each of them find the length $v(I_i)$ of the longest *chain* $I_i \subset I_{i_1} \subset I_{i_2} \subset \ldots$.



Statistics: 113 submissions, 42 accepted, 36 unknown

G: Going in Circles

Problem Author: Timon Knigge



Problem

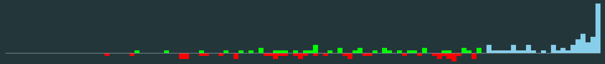
Determine the number n of train carriages of a circular train using at most $3n + 500$ steps.

In each step, you can either:

- move one carriage to the left,
- move one carriage to the right, or
- toggle the light switch in the current carriage.

G: Going in Circles

Problem Author: Timon Knigge



Problem

Determine the number n of train carriages of a circular train using at most $3n + 500$ steps.
In each step, you can either:

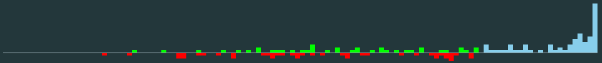
- move one carriage to the left,
- move one carriage to the right, or
- toggle the light switch in the current carriage.

Solution

- Naive solution: for some x, walk x steps to the right turning everything off, then flip one light switch, and walk x steps back to see if the light changed somewhere.

G: Going in Circles

Problem Author: Timon Knigge



Problem

Determine the number n of train carriages of a circular train using at most $3n + 500$ steps.
In each step, you can either:

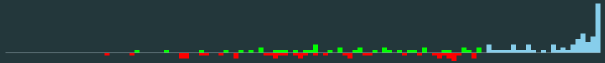
- move one carriage to the left,
- move one carriage to the right, or
- toggle the light switch in the current carriage.

Solution

- Naive solution: for some x, walk x steps to the right turning everything off, then flip one light switch, and walk x steps back to see if the light changed somewhere.
- If it did, then you know the length. If not, then try again with a larger x.

G: Going in Circles

Problem Author: Timon Knigge



Problem

Determine the number n of train carriages of a circular train using at most $3n + 500$ steps.
In each step, you can either:

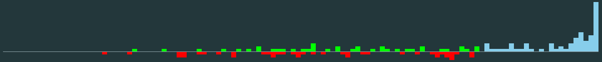
- move one carriage to the left,
- move one carriage to the right, or
- toggle the light switch in the current carriage.

Solution

- Naive solution: for some x, walk x steps to the right turning everything off, then flip one light switch, and walk x steps back to see if the light changed somewhere.
- If it did, then you know the length. If not, then try again with a larger x.
- This does not work: for small x, there is a lot of repetition so you need too many queries if n is large. For large x, you use too many queries if n is small.

G: Going in Circles

Problem Author: Timon Knigge



Problem

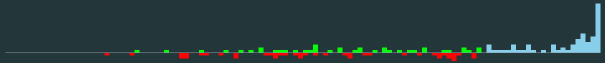
Determine the number n of train carriages of a circular train using at most $3n + 500$ steps.

Solution

- Alternative solution: use randomization.

G: Going in Circles

Problem Author: Timon Knigge



Problem

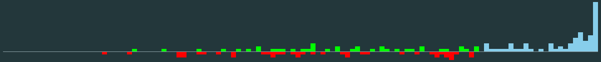
Determine the number n of train carriages of a circular train using at most $3n + 500$ steps.

Solution

- Alternative solution: use randomization.
- Choose a random sequence of bits of sufficient size (e.g. 25).

G: Going in Circles

Problem Author: Timon Knigge



Problem

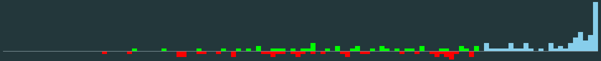
Determine the number n of train carriages of a circular train using at most $3n + 500$ steps.

Solution

- Alternative solution: use randomization.
- Choose a random sequence of bits of sufficient size (e.g. 25).
- Set the initial 25 bits to the chosen sequence.

G: Going in Circles

Problem Author: Timon Knigge



Problem

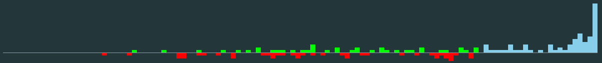
Determine the number n of train carriages of a circular train using at most $3n + 500$ steps.

Solution

- Alternative solution: use randomization.
- Choose a random sequence of bits of sufficient size (e.g. 25).
- Set the initial 25 bits to the chosen sequence.
- Walk to the right and keep track of the last read 25 bits.

G: Going in Circles

Problem Author: Timon Knigge



Problem

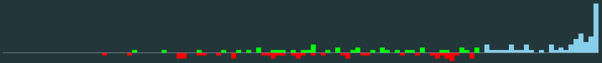
Determine the number n of train carriages of a circular train using at most $3n + 500$ steps.

Solution

- Alternative solution: use randomization.
- Choose a random sequence of bits of sufficient size (e.g. 25).
- Set the initial 25 bits to the chosen sequence.
- Walk to the right and keep track of the last read 25 bits.
- If the last read bits correspond to the chosen sequence, we assume we made a full round.

G: Going in Circles

Problem Author: Timon Knigge



Problem

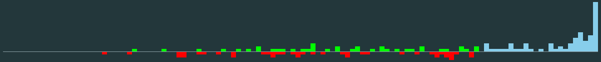
Determine the number n of train carriages of a circular train using at most $3n + 500$ steps.

Solution

- Alternative solution: use randomization.
- Choose a random sequence of bits of sufficient size (e.g. 25).
- Set the initial 25 bits to the chosen sequence.
- Walk to the right and keep track of the last read 25 bits.
- If the last read bits correspond to the chosen sequence, we assume we made a full round.
- Determine the length of the round using the number of steps made.

G: Going in Circles

Problem Author: Timon Knigge



Problem

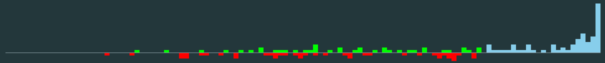
Determine the number n of train carriages of a circular train using at most $3n + 500$ steps.

Pitfalls

- The chosen bit sequence is not sufficiently long: De Bruijn sequences cover all 15-bit patterns..

G: Going in Circles

Problem Author: Timon Knigge



Problem

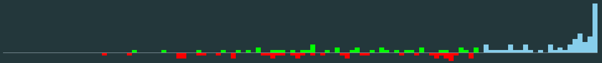
Determine the number n of train carriages of a circular train using at most $3n + 500$ steps.

Pitfalls

- The chosen bit sequence is not sufficiently long: De Bruijn sequences cover all 15-bit patterns..
- The chosen bit sequence is not “sufficiently random”:
  - $0000\ldots$, $010101\ldots$,
  - the default output of `rand()`,
  - the binary representation of special numbers: π, e, $\pi/2$, ϕ.

G: Going in Circles

Problem Author: Timon Knigge



Problem

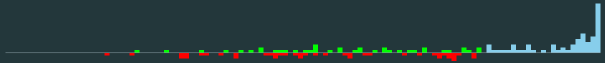
Determine the number n of train carriages of a circular train using at most $3n + 500$ steps.

Pitfalls

- The chosen bit sequence is not sufficiently long: De Bruijn sequences cover all 15-bit patterns..
- The chosen bit sequence is not “sufficiently random”:
  - $0000\ldots$, $010101\ldots$,
  - the default output of `rand()`,
  - the binary representation of special numbers: π, e, $\pi/2$, ϕ.
- Carefully handle the case where n is smaller than the length of the chosen sequence!

G: Going in Circles

Problem Author: Timon Knigge



Problem

Determine the number n of train carriages of a circular train using at most $3n + 500$ steps.

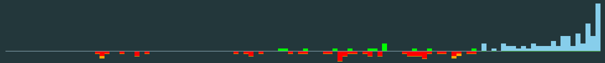
Pitfalls

- The chosen bit sequence is not sufficiently long: De Bruijn sequences cover all 15-bit patterns..
- The chosen bit sequence is not "sufficiently random":
  - $0000\ldots$, $010101\ldots$,
  - the default output of `rand()`,
  - the binary representation of special numbers: π, e, $\pi/2$, ϕ.
- Carefully handle the case where n is smaller than the length of the chosen sequence!

Statistics: 148 submissions, 39 accepted, 67 unknown

K: Kebab Pizza

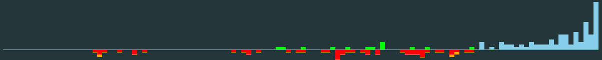
Problem Author: Wendy Yi



Problem

Spread a number of pizza toppings around a circular pizza such that:

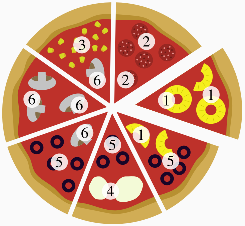
- each pizza topping only appears on some consecutive segment of the slices,
- there are at most two toppings on each slice, and
- the topping combinations match with a given list of preferences.

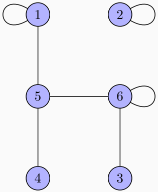


Problem

Spread a number of pizza toppings around a circular pizza such that:

- each pizza topping only appears on some consecutive segment of the slices,
- there are at most two toppings on each slice, and
- the topping combinations match with a given list of preferences.



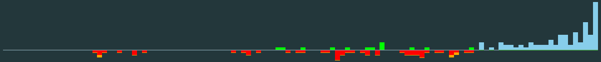


Insight

Model the problem as a graph, with the toppings as nodes and the topping combinations as edges.

K: Kebab Pizza

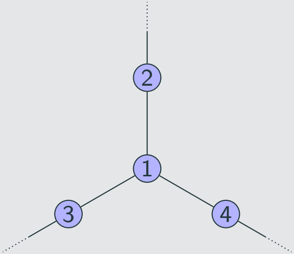
Problem Author: Wendy Yi



Solution

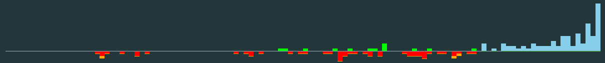
If any node has at least 3 non-leaf neighbours, then the answer is `impossible`:

- Suppose 1 has neighbours 2, 3 and 4, which each have a neighbour other than 1.



K: Kebab Pizza

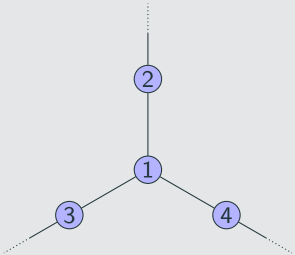
Problem Author: Wendy Yi

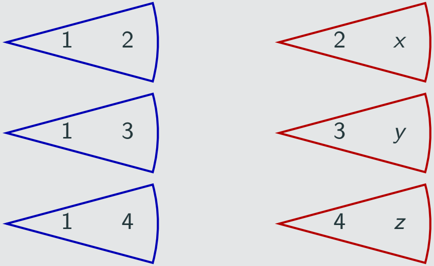


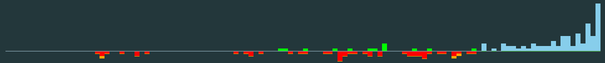
Solution

If any node has at least 3 non-leaf neighbours, then the answer is `impossible`:

- Suppose 1 has neighbours 2, 3 and 4, which each have a neighbour other than 1.
- There are slices $(1, 2)$, $(1, 3)$, $(1, 4)$ and $(2, x)$, $(3, y)$, $(4, z)$ with $1 \notin \{x, y, z\}$.



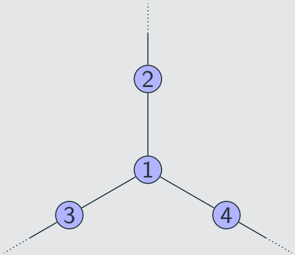


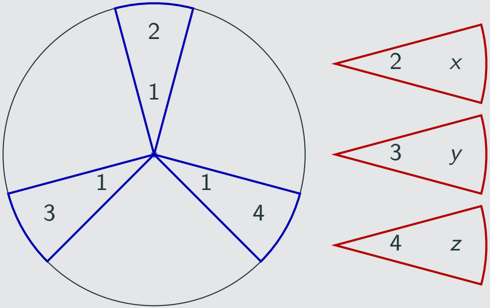


Solution

If any node has at least 3 non-leaf neighbours, then the answer is `impossible`:

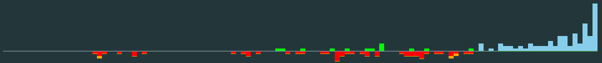
- Suppose 1 has neighbours 2, 3 and 4, which each have a neighbour other than 1.
- There are slices $(1,2)$, $(1,3)$, $(1,4)$ and $(2,x)$, $(3,y)$, $(4,z)$ with $1 \notin \{x,y,z\}$.
- Place the slices $(1,2)$, $(1,3)$, $(1,4)$ somewhere on the pizza.





K: Kebab Pizza

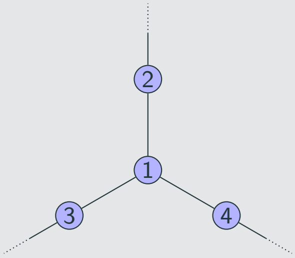
Problem Author: Wendy Yi

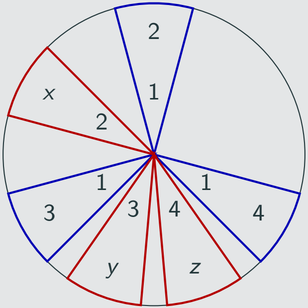


Solution

If any node has at least 3 non-leaf neighbours, then the answer is `impossible`:

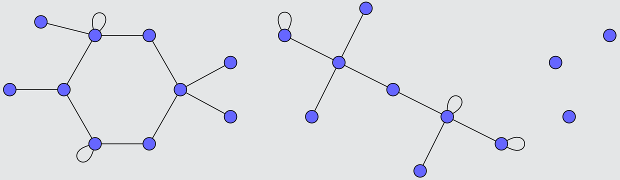
- Suppose 1 has neighbours 2, 3 and 4, which each have a neighbour other than 1.
- There are slices $(1,2)$, $(1,3)$, $(1,4)$ and $(2,x)$, $(3,y)$, $(4,z)$ with $1 \notin \{x,y,z\}$.
- Place the slices $(1,2)$, $(1,3)$, $(1,4)$ somewhere on the pizza.
- Slices $(2,x)$, $(3,y)$, $(4,z)$ go somewhere between these $\rightsquigarrow$ no consecutive range of 1's possible.

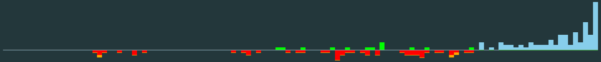




Solution

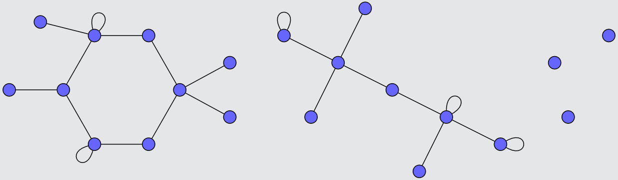
Otherwise, the graph consists of cycles and paths, possibly with extra leaves and self loops:



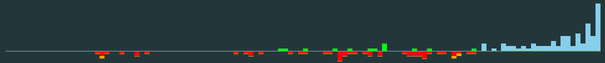


Solution

Otherwise, the graph consists of cycles and paths, possibly with extra leaves and self loops:

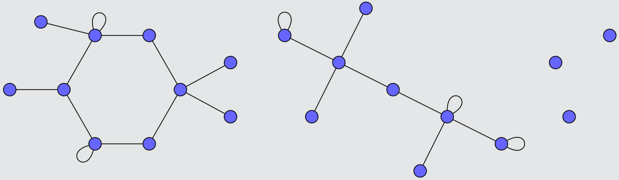


- Find all components and determine whether they are cycles or paths, e.g. by counting nodes, edges and loops.

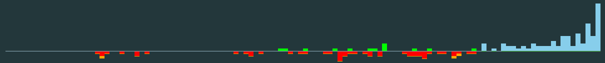


Solution

Otherwise, the graph consists of cycles and paths, possibly with extra leaves and self loops:

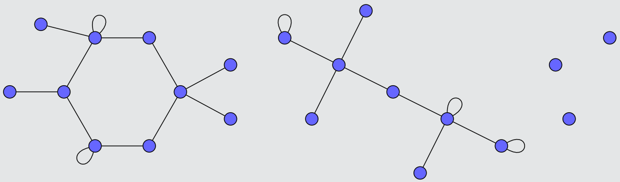


- Find all components and determine whether they are cycles or paths, e.g. by counting nodes, edges and loops.
- Remove all degree 0 vertices, corresponding to toppings not wanted by anybody.

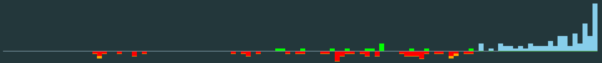


Solution

Otherwise, the graph consists of cycles and paths, possibly with extra leaves and self loops:

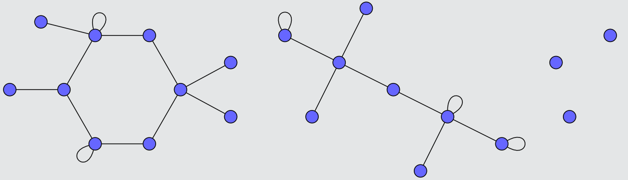


- Find all components and determine whether they are cycles or paths, e.g. by counting nodes, edges and loops.
- Remove all degree 0 vertices, corresponding to toppings not wanted by anybody.
- The answer is `possible` iff the graph is connected or all components are paths.



Solution

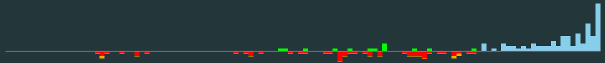
Otherwise, the graph consists of cycles and paths, possibly with extra leaves and self loops:



- Find all components and determine whether they are cycles or paths, e.g. by counting nodes, edges and loops.
- Remove all degree 0 vertices, corresponding to toppings not wanted by anybody.
- The answer is `possible` iff the graph is connected or all components are paths.
- Potential pitfalls: isolated vertices, paths of length 1, cycles of length 1, duplicate edges. . .

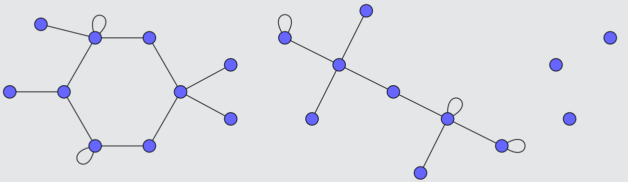
K: Kebab Pizza

Problem Author: Wendy Yi



Solution

Otherwise, the graph consists of cycles and paths, possibly with extra leaves and self loops:



- Find all components and determine whether they are cycles or paths, e.g. by counting nodes, edges and loops.
- Remove all degree 0 vertices, corresponding to toppings not wanted by anybody.
- The answer is `possible` iff the graph is connected or all components are paths.
- Potential pitfalls: isolated vertices, paths of length 1, cycles of length 1, duplicate edges. . .

Statistics: 155 submissions, 13 accepted, 90 unknown

L: Last Guess

Problem Author: Paul Wild

Problem

Given a game of Wordle with a word of length ℓ and g guesses with $g - 1$ guesses already made, find a valid final guess.

L: Last Guess

Problem Author: Paul Wild

Problem

Given a game of Wordle with a word of length ℓ and g guesses with $g-1$ guesses already made, find a valid final guess.

Observations

- What can we learn from an existing guess?

L: Last Guess

Problem Author: Paul Wild

Problem

Given a game of Wordle with a word of length ℓ and g guesses with $g-1$ guesses already made, find a valid final guess.

Observations

- What can we learn from an existing guess?
  - Green position: given letter is at that position.

L: Last Guess

Problem Author: Paul Wild

Problem

Given a game of Wordle with a word of length ℓ and g guesses with $g-1$ guesses already made, find a valid final guess.

Observations

- What can we learn from an existing guess?
  - Green position: given letter is at that position.
  - Yellow or gray position: given letter is not at that position.

L: Last Guess

Problem Author: Paul Wild

Problem

Given a game of Wordle with a word of length ℓ and g guesses with $g-1$ guesses already made, find a valid final guess.

Observations

- What can we learn from an existing guess?
  - Green position: given letter is at that position.
  - Yellow or gray position: given letter is not at that position.
  - A letter appears in the solution at least as often as the maximum number of green + yellow positions.

L: Last Guess

Problem Author: Paul Wild

Problem

Given a game of Wordle with a word of length ℓ and g guesses with $g-1$ guesses already made, find a valid final guess.

Observations

- What can we learn from an existing guess?
  - Green position: given letter is at that position.
  - Yellow or gray position: given letter is not at that position.
  - A letter appears in the solution at least as often as the maximum number of green + yellow positions.
  - If that letter also appears in a gray position, it appears in the solution exactly as often as the maximum number of green + yellow positions.

L: Last Guess

Problem Author: Paul Wild

Problem

Given a game of Wordle with a word of length ℓ and g guesses with $g-1$ guesses already made, find a valid final guess.

Observations

- What can we learn from an existing guess?
  - Green position: given letter is at that position.
  - Yellow or gray position: given letter is not at that position.
  - A letter appears in the solution at least as often as the maximum number of green + yellow positions.
  - If that letter also appears in a gray position, it appears in the solution exactly as often as the maximum number of green + yellow positions.
- So, for each letter, we have
  - a list of positions in which it *must* appear,
  - a list of positions in which it *must not* appear, and
  - a lower and upper bound on the number of appearances.

L: Last Guess

Problem Author: Paul Wild

Problem

Given a game of Wordle with a word of length ℓ and g guesses with $g-1$ guesses already made, find a valid final guess.

Observations

- What can we learn from an existing guess?
  - Green position: given letter is at that position.
  - Yellow or gray position: given letter is not at that position.
  - A letter appears in the solution at least as often as the maximum number of green + yellow positions.
  - If that letter also appears in a gray position, it appears in the solution exactly as often as the maximum number of green + yellow positions.
- So, for each letter, we have
  - a list of positions in which it *must* appear,
  - a list of positions in which it *must not* appear, and
  - a lower and upper bound on the number of appearances.
- How to find a word satisfying these requirements?

L: Last Guess

Problem Author: Paul Wild

Observations

- For each letter ℓ, we have
  - a list of positions in which it *must* appear,
  - a list of positions in which it *must not* appear, and
  - a lower bound l_ℓ and upper bound u_ℓ on the number of appearances.

Solution

L: Last Guess

Problem Author: Paul Wild

Observations

- For each letter ℓ, we have
  - a list of positions in which it *must* appear,
  - a list of positions in which it *must not* appear, and
  - a lower bound l_ℓ and upper bound u_ℓ on the number of appearances.

Solution

- First, consider simplified version where $l_\ell = 0$ for all ℓ.

L: Last Guess

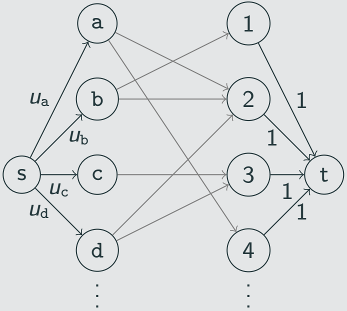
Problem Author: Paul Wild

Observations

- For each letter ℓ, we have
  - a list of positions in which it *must* appear,
  - a list of positions in which it *must not* appear, and
  - a lower bound l_ℓ and upper bound u_ℓ on the number of appearances.

Solution

- First, consider simplified version where $l_\ell = 0$ for all ℓ.
- Solvable using max-flow
  - Green positions: single incoming edge.
  - Otherwise: incoming edge for every possible character.



L: Last Guess

Problem Author: Paul Wild

Solution

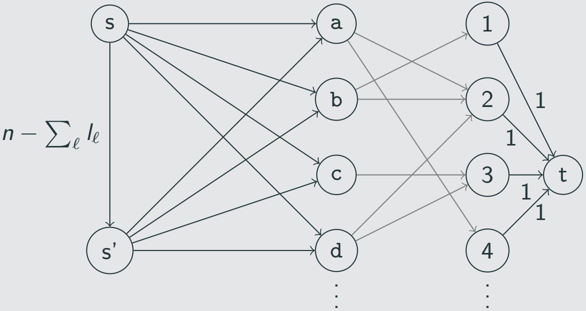
- Multiple ways to extend this to arbitrary lower bounds.

Solution

- Multiple ways to extend this to arbitrary lower bounds.
  - If you have the code: min-cost max-flow.

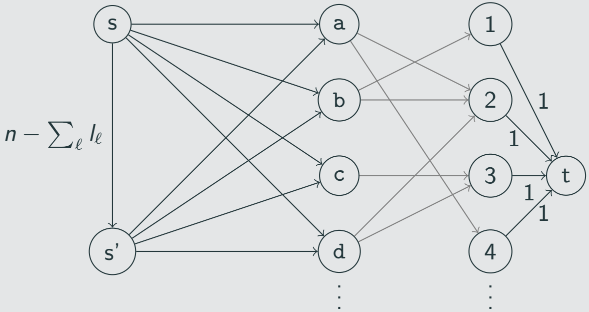
Solution

- Multiple ways to extend this to arbitrary lower bounds.
  - If you have the code: min-cost max-flow.
- Also possible: more clever max-flow modelling.
  - Edge from s to a letter ℓ has capacity l_ℓ;
  - Edge from s' to a letter ℓ has capacity $u_\ell - l_\ell$.



Solution

- Multiple ways to extend this to arbitrary lower bounds.
  - If you have the code: min-cost max-flow.
- Also possible: more clever max-flow modelling.
  - Edge from s to a letter ℓ has capacity l_ℓ;
  - Edge from s' to a letter ℓ has capacity $u_\ell - l_\ell$.



Statistics: 82 submissions, 4 accepted, 57 unknown

A: Alternating Algorithm

Problem Author: Bjarki Ágúst Guðmundsson

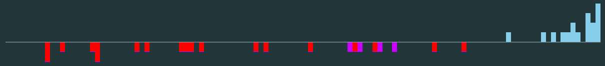
Problem

Given integers a_0 to a_n, how many of the following iterations does it take to sort them:

- Odd rounds: Sort pairs (a_0, a_1), (a_2, a_3), $\ldots$.
- Even rounds: Sort pairs (a_1, a_2), (a_3, a_4), $\ldots$.

A: Alternating Algorithm

Problem Author: Bjarki Ágúst Guðmundsson



Problem

Given integers a_0 to a_n, how many of the following iterations does it take to sort them:

- Odd rounds: Sort pairs (a_0, a_1), (a_2, a_3), $\ldots$.
- Even rounds: Sort pairs (a_1, a_2), (a_3, a_4), $\ldots$.

Observations

- Naive solution: simulating $\mathcal{O}(n)$ steps takes $\mathcal{O}(n^2)$ time.

Problem

Given integers a_0 to a_n, how many of the following iterations does it take to sort them:

- Odd rounds: Sort pairs (a_0, a_1), (a_2, a_3), $\ldots$.
- Even rounds: Sort pairs (a_1, a_2), (a_3, a_4), $\ldots$.

Observations

- Naive solution: simulating $\mathcal{O}(n)$ steps takes $\mathcal{O}(n^2)$ time.
- Say the last swap is (x, y).

A: Alternating Algorithm

Problem Author: Bjarki Ágúst Guðmundsson

Problem

Given integers a_0 to a_n, how many of the following iterations does it take to sort them:

- Odd rounds: Sort pairs (a_0, a_1), (a_2, a_3), $\ldots$.
- Even rounds: Sort pairs (a_1, a_2), (a_3, a_4), $\ldots$.

Observations

- Naive solution: simulating $\mathcal{O}(n)$ steps takes $\mathcal{O}(n^2)$ time.
- Say the last swap is (x, y).
- Replacing $a_i \leq x$ by 0 and $a_i > x$ by 1 gives an input that takes the same number of iterations.

Problem

Given integers a_0 to a_n, how many of the following iterations does it take to sort them:

- Odd rounds: Sort pairs (a_0, a_1), (a_2, a_3), ...
- Even rounds: Sort pairs (a_1, a_2), (a_3, a_4), ...

Observations

- Naive solution: simulating $\mathcal{O}(n)$ steps takes $\mathcal{O}(n^2)$ time.
- Say the last swap is (x, y).
- Replacing $a_i \leq x$ by 0 and $a_i > x$ by 1 gives an input that takes the same number of iterations.
- Idea: incrementally solve this 01-instance for every $x = a_i$ and take the maximum.

A: Alternating Algorithm

Problem Author: Bjarki Ágúst Guðmundsson

```
...X.XXX.X.XXX..
..X.X.XXX.X.XX..
.X.X.X.XXX.X.X..
X.X.X.X.XXX.X...
XX.X.X.X.XXX....
XXX.X.X.X.XX....
XXXX.X.X.X.X....
XXXXX.X.X.X.....
XXXXXX.X.X......
XXXXXXX.X.......
XXXXXXXX........
```

Solution for 01-instance

- 0s move left, 1s move right.

```
...X.XXX.X.XXX..
..X.X.XXX.X.XX..
.X.X.X.XXX.X.X..
X.X.X.X.XXX.X...
XX.X.X.X.XXX....
XXX.X.X.X.XX....
XXXX.X.X.X.X....
XXXXX.X.X.X.....
XXXXXX.X.X......
XXXXXXX.X.......
XXXXXXXX........
```

Solution for 01-instance

- 0s move left, 1s move right.
- The rightmost 0 is the last 0 to be *fixed*.

A: Alternating Algorithm

Problem Author: Bjarki Ágúst Guðmundsson

```
...X.XXX.X.XXX..
..X.X.XXX.X.XX..
.X.X.X.XXX.X.X..
X.X.X.X.XXX.X...
XX.X.X.X.XXX....
XXX.X.X.X.XX....
XXXX.X.X.X.X....
XXXXX.X.X.X.....
XXXXXX.X.X......
XXXXXXX.X.......
XXXXXXXX........
```

Solution for 01-instance

- 0s move left, 1s move right.
- The rightmost 0 is the last 0 to be *fixed*.
- A 0 is *congested* when blocked by another 0.

A: Alternating Algorithm

Problem Author: Bjarki Ágúst Guðmundsson

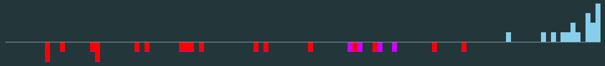
```
...X.XXX.X.XXX..
..X.X.XXX.X.XX..
.X.X.X.XXX.X.X..
X.X.X.X.XXX.X...
XX.X.X.X.XXX....
XXX.X.X.X.XX....
XXXX.X.X.X.X....
XXXXX.X.X.X.....
XXXXXX.X.X......
XXXXXXX.X.......
XXXXXXXX........
```

Solution for 01-instance

- 0s move left, 1s move right.
- The rightmost 0 is the last 0 to be *fixed*.
- A 0 is *congested* when blocked by another 0.
- For each unfixed 0, the total time is at least:
  - The time to fix this 0 fixed assuming no congestion, plus
  - the number of 0s after it, since at most one 0 can be fixed in each iteration.

A: Alternating Algorithm

Problem Author: Bjarki Ágúst Guðmundsson



```
...X.XXX.X.XXX..
..X.X.XXX.X.XX..
.X.X.X.XXX.X.X..
X.X.X.X.XXX.X...
XX.X.X.X.XXX....
XXX.X.X.X.XX....
XXXX.X.X.X.X....
XXXXX.X.X.X.....
XXXXXX.X.X......
XXXXXXX.X.......
XXXXXXXX........
```

Solution for 01-instance

- 0s move left, 1s move right.
- The rightmost 0 is the last 0 to be *fixed*.
- A 0 is *congested* when blocked by another 0.
- For each unfixed 0, the total time is at least:
  - The time to fix this 0 fixed assuming no congestion, plus
  - the number of 0s after it, since at most one 0 can be fixed in each iteration.
- The maximum over all unfixed 0s is the answer.

A: Alternating Algorithm

Problem Author: Bjarki Ágúst Guðmundsson

Problem

Given integers a_0 to a_n, how many of the following iterations does it take to sort them:

- Odd rounds: Sort pairs (a_0, a_1), (a_2, a_3), $\ldots$.
- Even rounds: Sort pairs (a_1, a_2), (a_3, a_4), $\ldots$.

Solution

- Incrementally solve all 01-instances for increasing x.

[1]Check the README for details.

A: Alternating Algorithm

Problem Author: Bjarki Ágúst Guðmundsson

Problem

Given integers a_0 to a_n, how many of the following iterations does it take to sort them:

- Odd rounds: Sort pairs (a_0, a_1), (a_2, a_3), $\ldots$.
- Even rounds: Sort pairs (a_1, a_2), (a_3, a_4), $\ldots$.

Solution

- Incrementally solve all 01-instances for increasing x.
- Use a segment tree[1] to efficiently query the maximum time.

[1] Check the README for details.

A: Alternating Algorithm

Problem Author: Bjarki Ágúst Guðmundsson

Problem

Given integers a_0 to a_n, how many of the following iterations does it take to sort them:

- Odd rounds: Sort pairs (a_0, a_1), (a_2, a_3), $\ldots$.
- Even rounds: Sort pairs (a_1, a_2), (a_3, a_4), $\ldots$.

Solution

- Incrementally solve all 01-instances for increasing x.
- Use a segment tree[1] to efficiently query the maximum time.
- Incrementally update it for every 0 that changes to a 1.

[1] Check the README for details.

A: Alternating Algorithm

Problem Author: Bjarki Ágúst Guðmundsson

Problem

Given integers a_0 to a_n, how many of the following iterations does it take to sort them:

- Odd rounds: Sort pairs (a_0, a_1), (a_2, a_3), $\ldots$.
- Even rounds: Sort pairs (a_1, a_2), (a_3, a_4), $\ldots$.

Solution

- Incrementally solve all 01-instances for increasing x.
- Use a segment tree[1] to efficiently query the maximum time.
- Incrementally update it for every 0 that changes to a 1.

Statistics: 42 submissions, 0 accepted, 17 unknown

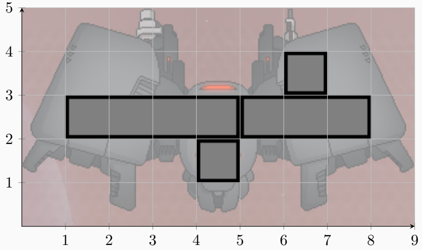
[1] Check the README for details.

F: Faster Than Light

Problem Author: Michael Zündorf

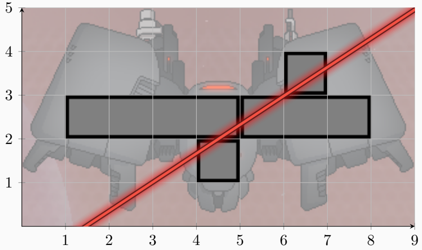
Problem

Given n axis-aligned rectangles, determine whether there is a line intersecting or touching all of them.



Problem

Given n axis-aligned rectangles, determine whether there is a line intersecting or touching all of them.



Observation

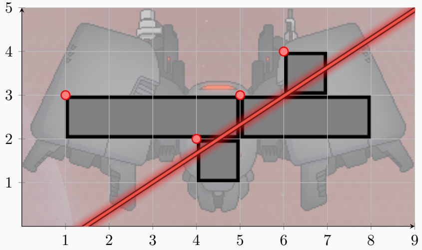
- If the solution line has positive slope, then:

F: Faster Than Light

Problem Author: Michael Zündorf

Problem

Given n axis-aligned rectangles, determine whether there is a line intersecting or touching all of them.



Observation

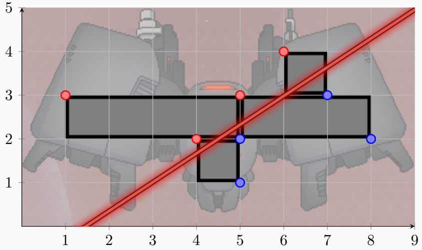
- If the solution line has positive slope, then:
  - it passes below the top left corners of every rectangle,

F: Faster Than Light

Problem Author: Michael Zündorf

Problem

Given n axis-aligned rectangles, determine whether there is a line intersecting or touching all of them.



Observation

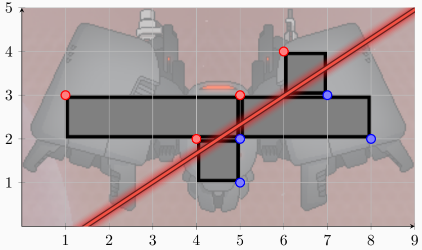
- If the solution line has positive slope, then:
  - it passes below the top left corners of every rectangle,
  - it passes over the bottom right corner of every rectangle.

F: Faster Than Light

Problem Author: Michael Zündorf

Problem

Given n axis-aligned rectangles, determine whether there is a line intersecting or touching all of them.

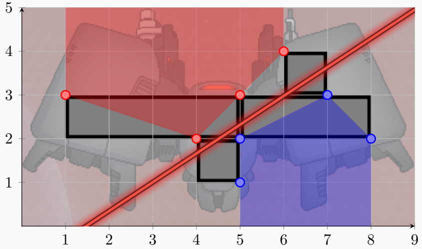


Observation

- If the solution line has positive slope, then:
  - it passes below the top left corners of every rectangle,
  - it passes over the bottom right corner of every rectangle.
- For lines with negative slope, something similar holds.

Problem

Given n axis-aligned rectangles, determine whether there is a line intersecting or touching all of them.



Observation

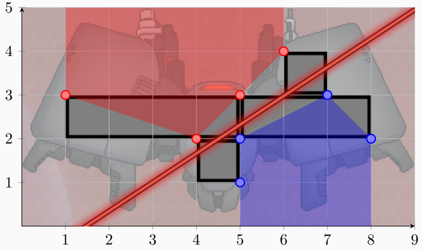
Use the upper convex hull of the red points and lower convex hull of the blue points.

F: Faster Than Light

Problem Author: Michael Zündorf

Problem

Given n axis-aligned rectangles, determine whether there is a line intersecting or touching all of them.



Observation

Use the upper convex hull of the red points and lower convex hull of the blue points.

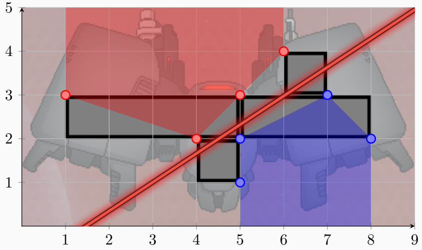
- A line that passes in between intersects all rectangles.

F: Faster Than Light

Problem Author: Michael Zündorf

Problem

Given n axis-aligned rectangles, determine whether there is a line intersecting or touching all of them.



Observation

Use the upper convex hull of the red points and lower convex hull of the blue points.

- A line that passes in between intersects all rectangles.
- A line inside a convex hull goes above/below a red/blue point.

F: Faster Than Light

Problem Author: Michael Zündorf

Problem

Given n axis-aligned rectangles, determine whether there is a line intersecting or touching all of them.

Solution

- First check for lines with positive slope:
  - Compute the lower convex hull of all top left corners.
  - Compute the upper convex hull of all bottom right corners.
  - Check (in linear time) whether these intersect.

F: Faster Than Light

Problem Author: Michael Zündorf

Problem

Given n axis-aligned rectangles, determine whether there is a line intersecting or touching all of them.

Solution

- First check for lines with positive slope:
  - Compute the lower convex hull of all top left corners.
  - Compute the upper convex hull of all bottom right corners.
  - Check (in linear time) whether these intersect.
- In a similar way, check for lines with negative slope.

F: Faster Than Light

Problem Author: Michael Zündorf

Problem

Given n axis-aligned rectangles, determine whether there is a line intersecting or touching all of them.

Solution

- First check for lines with positive slope:
  - Compute the lower convex hull of all top left corners.
  - Compute the upper convex hull of all bottom right corners.
  - Check (in linear time) whether these intersect.
- In a similar way, check for lines with negative slope.
- Also check for vertical lines.

F: Faster Than Light

Problem Author: Michael Zündorf

Problem

Given n axis-aligned rectangles, determine whether there is a line intersecting or touching all of them.

Solution

- First check for lines with positive slope:
  - Compute the lower convex hull of all top left corners.
  - Compute the upper convex hull of all bottom right corners.
  - Check (in linear time) whether these intersect.
- In a similar way, check for lines with negative slope.
- Also check for vertical lines.
- Total running time: $\mathcal{O}(n \log n)$.

F: Faster Than Light

Problem Author: Michael Zündorf

Problem

Given n axis-aligned rectangles, determine whether there is a line intersecting or touching all of them.

Solution

- First check for lines with positive slope:
  - Compute the lower convex hull of all top left corners.
  - Compute the upper convex hull of all bottom right corners.
  - Check (in linear time) whether these intersect.
- In a similar way, check for lines with negative slope.
- Also check for vertical lines.
- Total running time: $\mathcal{O}(n \log n)$.

Fun Fact

What is a laser?

F: Faster Than Light

Problem Author: Michael Zündorf

Problem

Given n axis-aligned rectangles, determine whether there is a line intersecting or touching all of them.

Solution

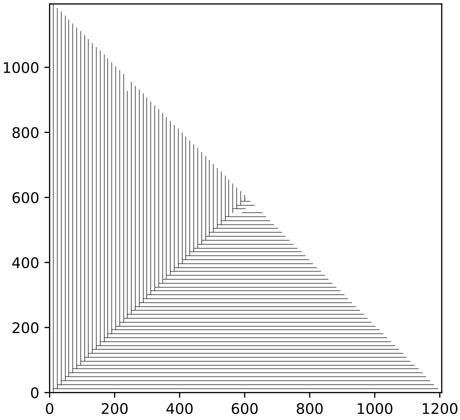
- First check for lines with positive slope:
  - Compute the lower convex hull of all top left corners.
  - Compute the upper convex hull of all bottom right corners.
  - Check (in linear time) whether these intersect.
- In a similar way, check for lines with negative slope.
- Also check for vertical lines.
- Total running time: $\mathcal{O}(n \log n)$.

Fun Fact

What is a laser? We only defined "hull beam".

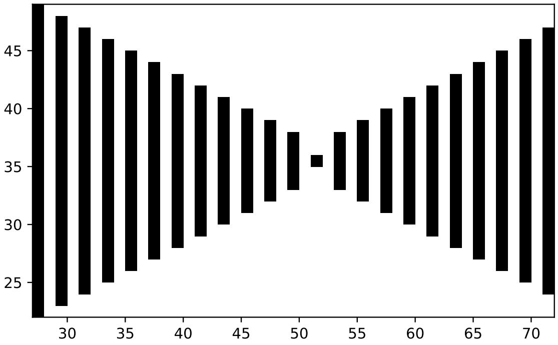
F: Faster Than Light

Problem Author: Michael Zündorf



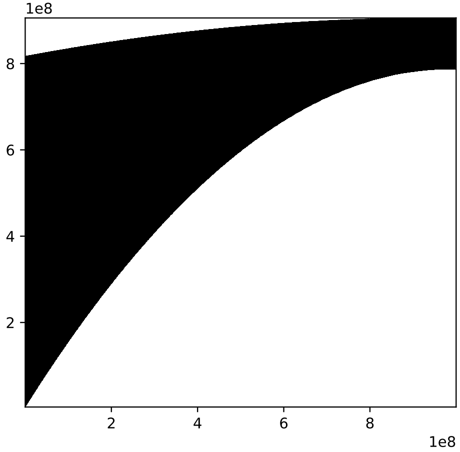
F: Faster Than Light

Problem Author: Michael Zündorf



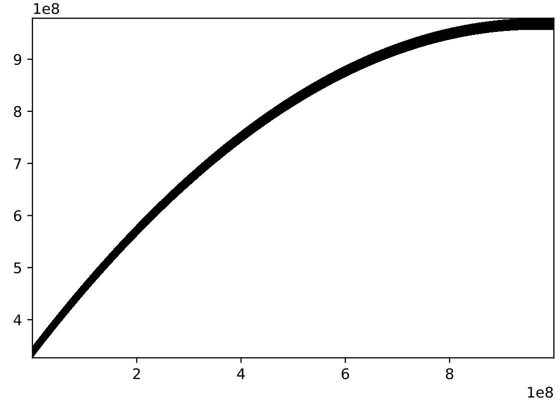
F: Faster Than Light

Problem Author: Michael Zündorf



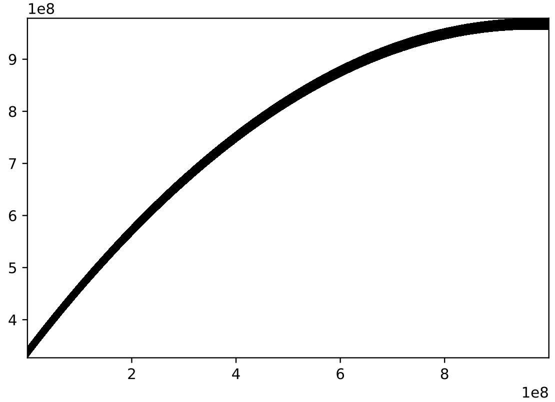
F: Faster Than Light

Problem Author: Michael Zündorf



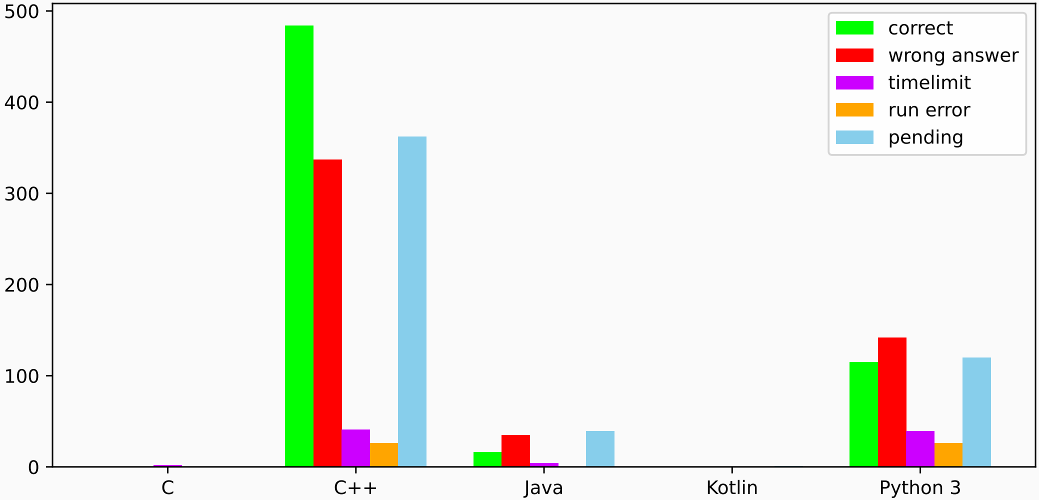
F: Faster Than Light

Problem Author: Michael Zündorf



Statistics: 29 submissions, 0 accepted, 26 unknown

Language stats



Random facts

Jury work

- 720 commits (including test session) (last year: 632)

Random facts

Jury work

- 720 commits (including test session) (last year: 632)
- 1424 secret test cases (last year: 681) ($118\frac{2}{3}$ per problem!)

Random facts

Jury work

- 720 commits (including test session) (last year: 632)
- 1424 secret test cases (last year: 681) ($118\frac{2}{3}$ per problem!)
- 239 jury solutions (last year: 248)

Random facts

Jury work

- 720 commits (including test session) (last year: 632)
- 1424 secret test cases (last year: 681) ($118\frac{2}{3}$ per problem!)
- 239 jury solutions (last year: 248)
- The minimum[2] number of lines the jury needed to solve all problems is

$$19 + 1 + 6 + 6 + 14 + 22 + 3 + 6 + 2 + 31 + 8 + 45 = 163$$

On average 13.6 lines per problem, down from 35.5 last year

[2] *After* code golfing

Random facts

Jury work

- 720 commits (including test session) (last year: 632)
- 1424 secret test cases (last year: 681) ($118\frac{2}{3}$ per problem!)
- 239 jury solutions (last year: 248)
- The minimum[2] number of lines the jury needed to solve all problems is

$$19 + 1 + 6 + 6 + 14 + 22 + 3 + 6 + 2 + 31 + 8 + 45 = 163$$

  On average 13.6 lines per problem, down from 35.5 last year

- Only team ORTEC beat us: they have a submission of 22 lines for Justice Served!

[2] *After* code golfing

Random facts

Jury dedication

- Most test cases for Faster Than Light were generated after midnight and/or yesterday.

Random facts

Jury dedication

- Most test cases for Faster Than Light were generated after midnight and/or yesterday.
- The 80–20 rule is a thing: 80% of our time is spent on 20% of the problem statement.

Random facts

Jury dedication

- Most test cases for Faster Than Light were generated after midnight and/or yesterday.
- The 80–20 rule is a thing: 80% of our time is spent on 20% of the problem statement.
- The longest discussions were about tiny style issues like “illustration” vs. “visualisation”.

Random facts