GCPC 2021
Presentation of solutions

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Jury and Testers

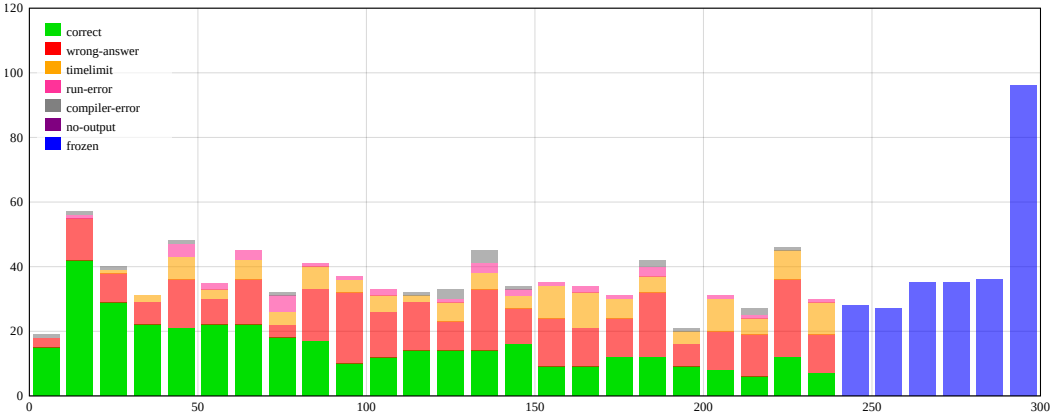
Thanks to the jury:

- Michael Baer (FAU)
- Gregor Behnke (Freiburg)
- Alexander Dietsch (FAU)
- Andreas Grigorjew (Helsinki)
- Florian Leimgruber (TUM)
- Felicia Lucke (CPUlm)
- Nathan Maier (CPUlm)
- Gregor Matl (TUM)
- Michael Ruderer (CPUlm)
- Gregor Schwarz (TUM)
- Paul Wild (FAU)
- Michael Zündorf (KIT)

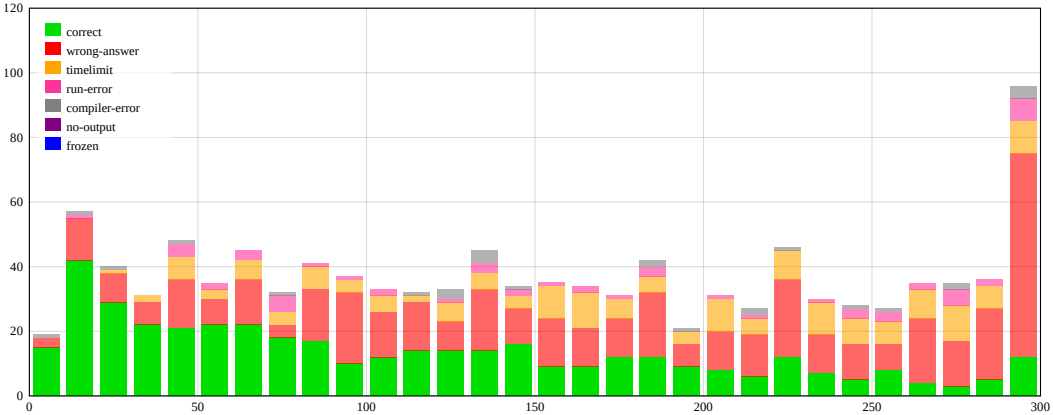
Thanks to our test reader:

- Philipp Reger (FAU)

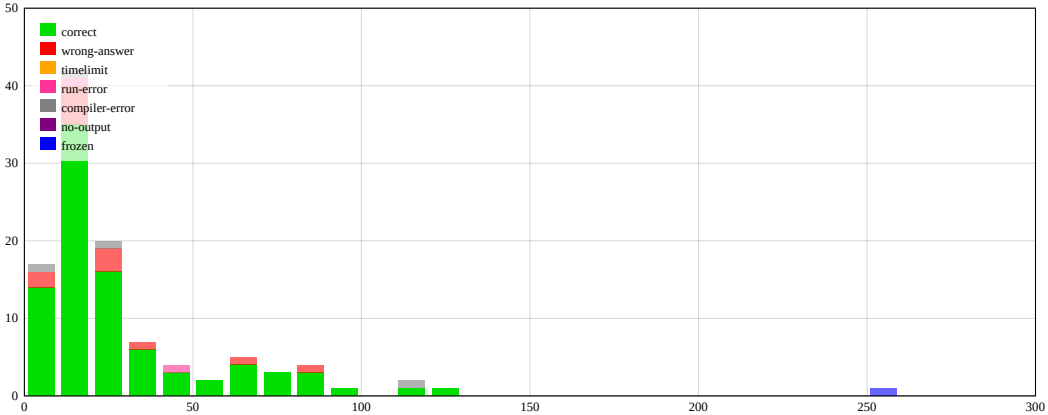
Statistics


correct
wrong-answer
timelimit
run-error
compiler-error
no-output
frozen
120
100
80
60
40
20
0
0
50
100
150
200
250
300

Statistics


correct
wrong-answer
timelimit
run-error
compiler-error
no-output
frozen
120
100
80
60
40
20
0
0
50
100
150
200
250
300

B – Brexiting and Brentering



B – Brexiting and Brentering

Problem

Given a name containing one or more vowels, cut everything off after the last vowel and instead append the suffix “ntry”.

Solution

In C++:

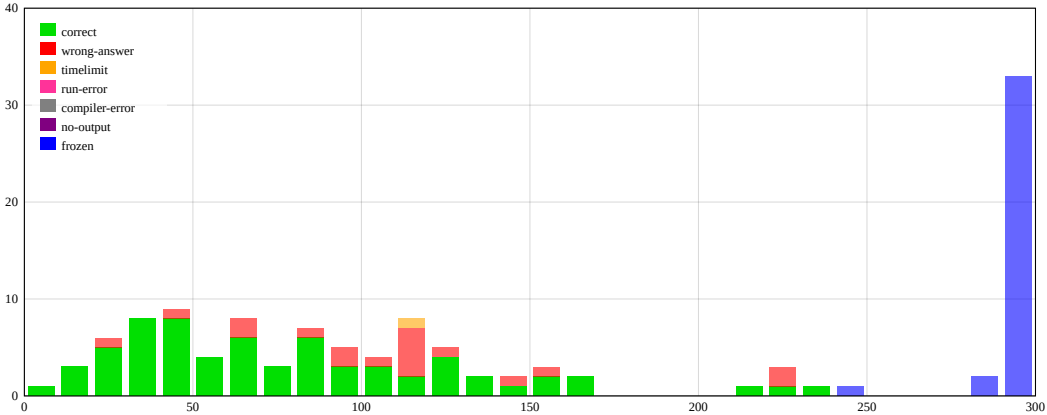
```
#include <bits/stdc++.h>
using namespace std;

int main() {
        string subject;
        cin >> subject;

        cout << subject.substr(0, subject.find_last_of("aeiou") + 1) << "ntry" << endl;
}
```

B – Brexiting and Brentering

Alternative solution (too late)

Don't do Brexit in the first place!

M – Monty's Hall



M – Monty's Hall

Problem

A generalised version of the Monty Hall problem: Given d doors with one correct one, you may select s. Then e unselected wrong doors are opened after which you may select again. What is the chance of the correct door being among your second selection?

M – Monty's Hall

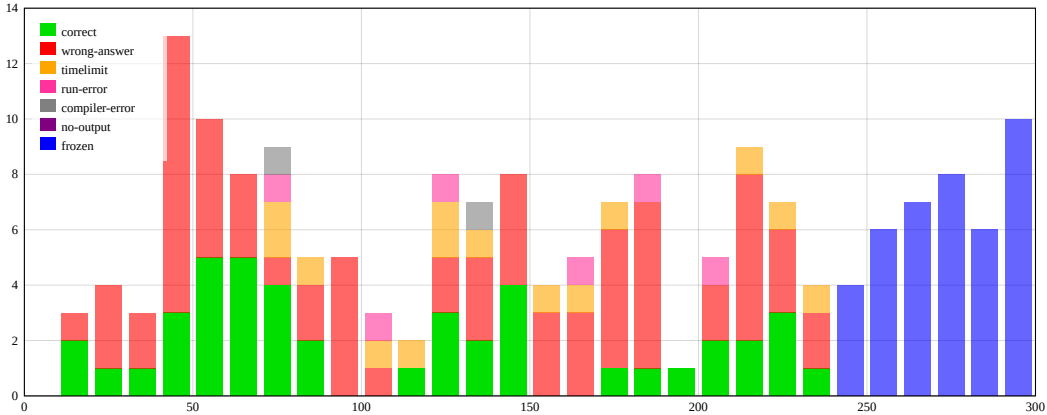
Problem

A generalised version of the Monty Hall problem: Given d doors with one correct one, you may select s. Then e unselected wrong doors are opened after which you may select again. What is the chance of the correct door being among your second selection?

Solution

- Observation: During the first selection you are less likely to hit the correct door than during the second. $\Rightarrow$ It is always better to change the selection (like in the original problem).
- Calculate the chance of selecting the correct door with your second selection.
- If there are fewer than s unselected doors during your second selection, keep a number of doors the same (let that number be x). Calculate how likely it is for the correct door to have been among these x doors during the first selection.

A – Amusement Arcade



A – Amusement Arcade

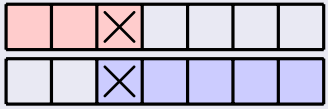
Problem

Given n arcades in a straight line and $(n+1)/2$ players who arrive one after another and always pick the most secluded arcade, where should the first player take a seat so that everyone can be seated and there is always one empty arcade between two neighbouring players?

A – Amusement Arcade

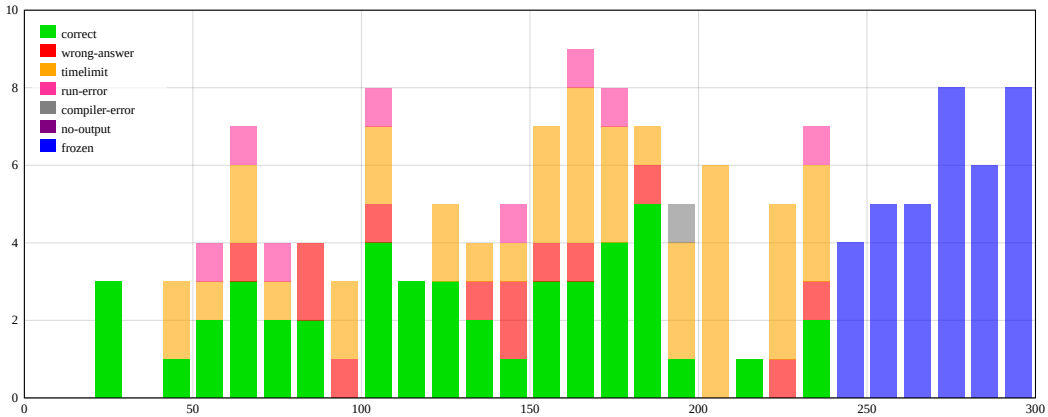
Solution

- Whenever $n = 2^k + 1$, the first player can take a seat on the very first or very last machine and a seating arrangement can be found.
- If the first player sits at machine x, she splits the interval in two parts and sits at the very end of both intervals.



- If the left and right interval can be expressed by $\ell = 2^a + 1$ and $r = 2^b + 1$, respectively, then a seating arrangement can be found.
- Check whether $n = 2^a + 2^b + 1$ either by
  - looping through all possible power-of-two-pairs (less than $4,000$) or
  - counting the ones in the binary representation of n.

H – Hectic Harbour II



H – Hectic Harbour II

Problem

There are two stacks with crates marked 0 to n. The crates 1 to n are moved away in that sequence. Before collecting a specific crate, all crates on top of it are moved to the other stack one-by-one first. How often is crate 0 on top of a stack after a crate has been collected?

H – Hectic Harbour II

Problem

There are two stacks with crates marked 0 to n. The crates 1 to n are moved away in that sequence. Before collecting a specific crate, all crates on top of it are moved to the other stack one-by-one first. How often is crate 0 on top of a stack after a crate has been collected?

Solution

- The two stacks can be reinterpreted as a doubly linked list:
  - Begin of the list is the bottom of the first stack.
  - End of the list the bottom of the second stack.
  - The top of the stacks is between two elements of the list.
- A removal step can be performed in $\mathcal{O}(1)$ by removing the element and setting the top of the stacks at its position.
- Simulate and count how often crate 0 is one of the two top elements.

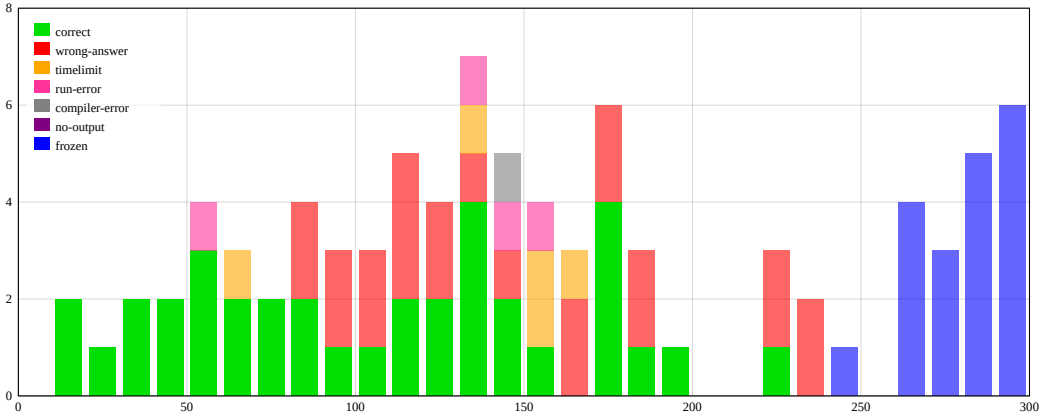
H – Hectic Harbour II

Alternative solution

- 0 can only be on top after the crate next to it was just removed.
- Starting from 0 iterate in one direction and repeatedly select the closest bigger element. Do that in both directions.
- The number of selected elements is the number of times 0 will be on top of a stack.

E.g. in 51203746 the elements 25 and 37 are selected $\Rightarrow$ 0 will be on top 4 times.

K – Killjoys' Conference



K – Killjoys' Conference

Problem

You are given a graph $G = (V, E)$. Count the number of *bipartitions*, i.e. partitions $A \uplus B = V$ such that no two vertices in A or B are adjacent.

K – Killjoys' Conference

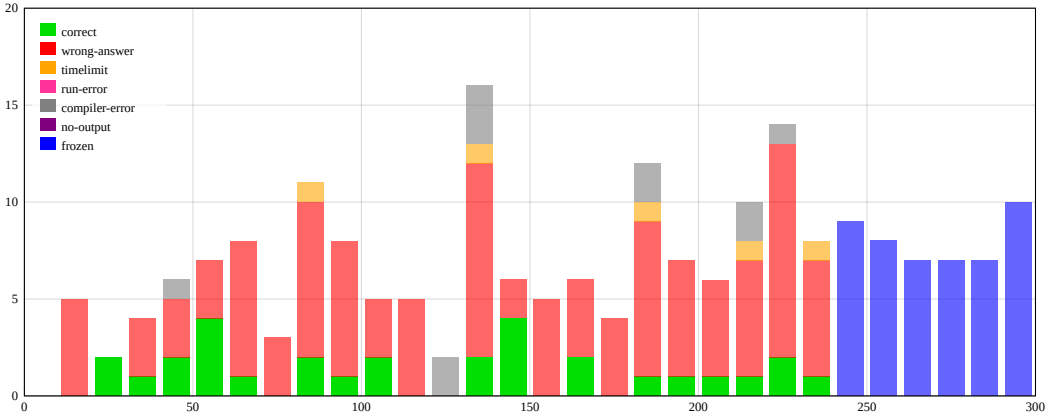
Problem

You are given a graph $G = (V, E)$. Count the number of *bipartitions*, i.e. partitions $A \uplus B = V$ such that no two vertices in A or B are adjacent.

Solution

- Consider the connected components $C_1, \ldots, C_k$ of G in isolation:
  - For each component, check if it has a bipartition using DFS.
  - Colour one vertex black, its neighbours white, its neighbours black and so on.
  - If two neighbours are given the same colour, output `impossible`.
- Within each component the colours may be swapped, so 2^k possibilities in total.
- One symmetry is remaining: (A, B) and (B, A) are considered the same partition.
- Final answer: $(2^{k-1} + 1) \mod p$

C – Card Trading



C – Card Trading

Problem

Given a number of buy orders and sell orders of a card, find the card price that generates the highest turnover.

C – Card Trading

Problem

Given a number of buy orders and sell orders of a card, find the card price that generates the highest turnover.

Solution

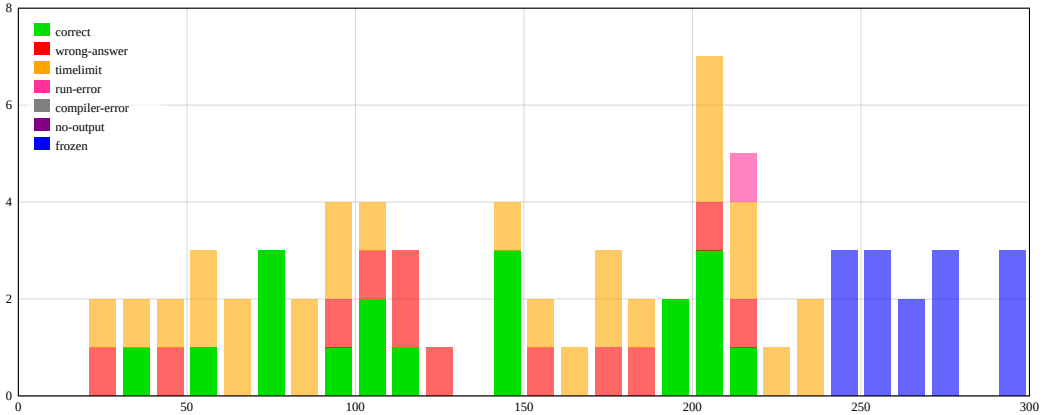
- Observation: The optimal card price is one of the prices in the input.
- A buyer is always willing to pay less $\rightarrow$ Starting from highest price, add all buyers to the next lower price.
- A seller is always willing to sell for more $\rightarrow$ Starting from the lowest price, add all sellers to the next higher price.
- Iterate over all prices and find highest turnover by matching the aggregated buyers and sellers.

C – Card Trading

Caveat

Double–precision floating point numbers are not accurate enough. Use extended precision floats instead. Alternatively turn the input into integers by multiply with 100 and make sure not to accidentally cast the output to a double–precision float when dividing by 100.

E – Excursion to Porvoo



E – Excursion to Porvoo

Problem

Given a DAG with nodes $x_1, \ldots, x_n$ and edges $x_i \to x_{i+1}$ with weight capacities, calculate the smallest $x_1 \to x_n$ path distance for each query g, using only edges of capacity $c \geq g$.

E – Excursion to Porvoo

Problem

Given a DAG with nodes $x_1, \ldots, x_n$ and edges $x_i \to x_{i+1}$ with weight capacities, calculate the smallest $x_1 \to x_n$ path distance for each query g, using only edges of capacity $c \geq g$.

Solution

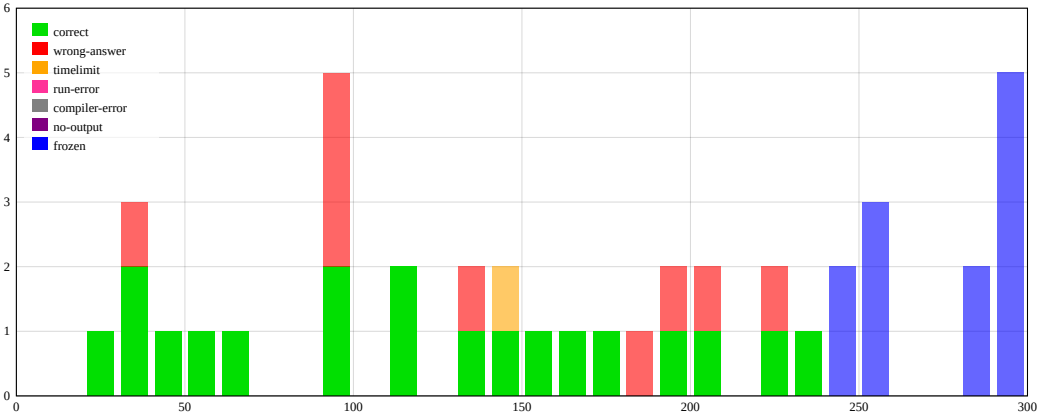
- By definition, a query g can traverse all edges a larger query $g' \geq g$ can traverse.
- Use a two pointer solution:
- Sort the queries and the edges by their capacities non-increasingly.

E – Excursion to Porvoo

Solution

- Iterare through the sorted queries g_i and add all the edges of capacity $c \geq g_i$ to the graph, which have not been added in prior iterations.
- Each time we add an edge, the fastest path might change, and we update the distance accordingly.
- Return *impossible* as long as x_n can not be reached from x_1.
- Runtime complexity: $\mathcal{O}(m \log m + q \log q)$.

G – Grid Delivery



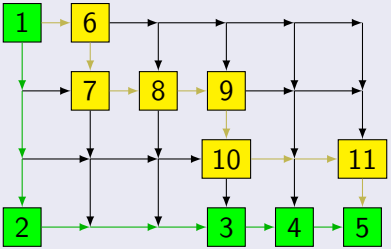
G – Grid Delivery

Problem

Given an $h \times w$ grid of streets where parcels are located at some of the intersections and you are only allowed to move south or east. How many trips, starting from the northwesternmost intersection and ending at the most southeasternmost intersection, do you have to take to collect all parcels?

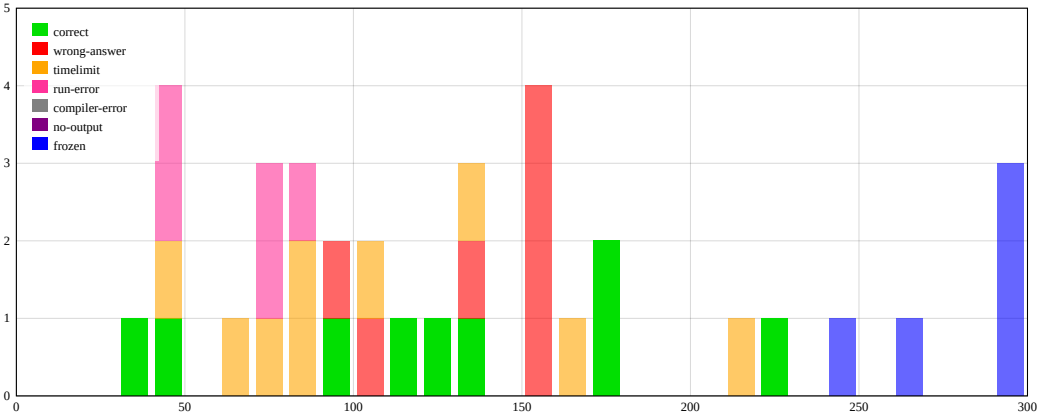
G – Grid Delivery

Solution



- Pickup the westernmost package reachable until you reach the southeasternmost intersection.
- Start over from nortwesternmost intersection.
- Keep track of which interesctions you already visited.
- Visiting each intersection at most once yields $\mathcal{O}(hw)$ runtime.

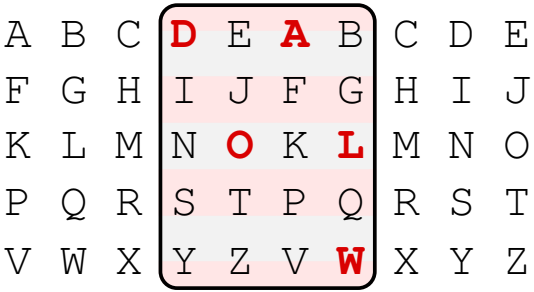
L – Looking for Waldo



L – Looking for Waldo

Problem

Given a $w \cdot h$ grid of letters, Find the smallest rectangle which contains at least one `W`,`A`,`L`,`D` and `O`.



L – Looking for Waldo

Problem

Given a $w \cdot h$ grid of letters, Find the smallest rectangle which contains at least one `W`,`A`,`L`,`D` and `O`.

Solution

- Let $next[x, y, c]$ describe the next occurrence of character c in row y after x.

	1	2	3	4	5	6	7	8	9
1	W	H	E	R	E	I	S	W	A

- $next[1, 1, W] = 1$
- $next[6, 1, W] = 8$
- $next[9, 1, W] = \text{inf}$

L – Looking for Waldo

Problem

Given a $w \cdot h$ grid of letters, Find the smallest rectangle which contains at least one `W`,`A`,`L`,`D` and `O`.

Solution

- Let $next[x, y, c]$ describe the next occurrence of character c in row y after x.
- Fix the upper left corner x, y of the rectangle and find the smallest rectangle with height $1, 2, \ldots$
  - For height 1 this is $\max(next[x, y, c])$ for $c \in W, A, L, D, O$.
  - For height 2 this is $\max(\min(next[x, y, c], next[x, y + 1, c]))$.
  - For height h', $\min(next[x, y, c], \ldots)$ can only decrease to $next[x, y + h' - 1, c]$.

  $\Rightarrow$ The inner minimums can be calculated in constant time. The outer maximum can be evaluated in constant time too since it only contains 5 expressions.

L – Looking for Waldo

Runtime

Why is this fast enough?

- There are $w \cdot h$ corners we can choose.
- The height can be up to h

$\Rightarrow$ The runtime is $\mathcal{O}(w \cdot h \cdot h)$. This sounds quadratic?

L – Looking for Waldo

Runtime

Why is this fast enough?

- There are $w \cdot h$ corners we can choose.
- The height can be up to h

$\Rightarrow$ The runtime is $\mathcal{O}(w \cdot h \cdot h)$. This sounds quadratic?

- Note that $n = w \cdot h$ was bounded in the input.
- If we assume $h \leq w$ the actual runtime is $\mathcal{O}(n\sqrt{n})$ which is fast enough.

L – Looking for Waldo

Runtime

Why is this fast enough?

- There are $w \cdot h$ corners we can choose.
- The height can be up to h

$\Rightarrow$ The runtime is $\mathcal{O}(w \cdot h \cdot h)$. This sounds quadratic?

- Note that $n = w \cdot h$ was bounded in the input.
- If we assume $h \leq w$ the actual runtime is $\mathcal{O}(n\sqrt{n})$ which is fast enough.
- If $h > w$ we can rotate or transpose the input to make $h \leq w$.

L – Looking for Waldo

Runtime

Why is this fast enough?

- There are $w \cdot h$ corners we can choose.
- The height can be up to h

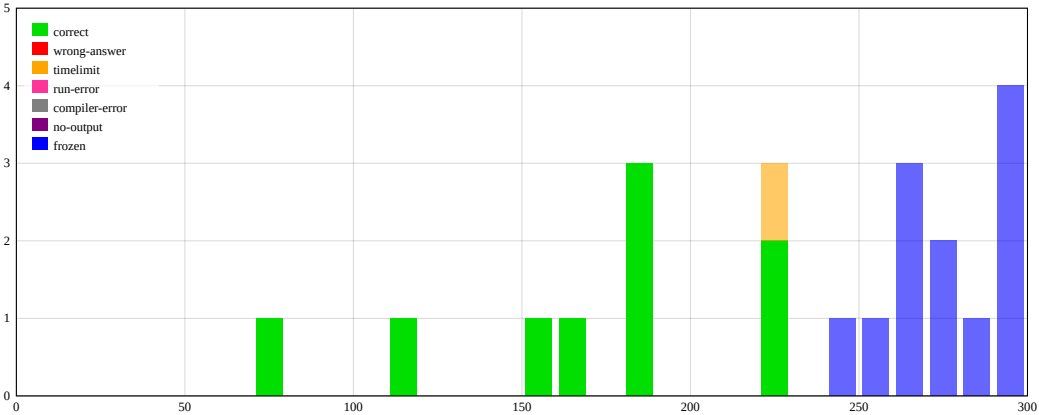
$\Rightarrow$ The runtime is $\mathcal{O}(w \cdot h \cdot h)$. This sounds quadratic?

- Note that $n = w \cdot h$ was bounded in the input.
- If we assume $h \leq w$ the actual runtime is $\mathcal{O}(n\sqrt{n})$ which is fast enough.
- If $h > w$ we can rotate or transpose the input to make $h \leq w$.

Alternatives

- A faster runtime can be achieved with a *Divide and Conquer* approach, which runs in $\mathcal{O}(n \log n)$.
- However, this was not required.

I – Index Case


correct
wrong-answer
timelimit
run-error
compiler-error
no-output
frozen
0
1
2
3
4
5
0
50
100
150
200
250
300

I – Index Case

Problem

Given a cyclic *cellular automaton* with state s and transition function f, check if a state x exists with $f(x) = s$.

I – Index Case

Problem

Given a cyclic *cellular automaton* with state s and transition function f, check if a state x exists with $f(x) = s$.

Solution

- First we choose a pair $(x[0], x[1])$
- Starting from $i = 1$ until n (where $i = n$ means $i = 0$)
  - We keep a list of possible states for the pair $(x[i-1], x[i])$ (just the chosen possibility when i=1)
  - The pair $(x[i+1], x[i])$ is possible iff $f(x[i-1], x[i], x[i+1]) = s[i]$
- Check if the initial tuple is in our list after the last step.

I – Index Case

Problem

Given a cyclic *cellular automaton* with state s and transition function f, check if a state x exists with $f(x) = s$.

Solution

- First we choose a pair $(x[0], x[1])$
- Starting from $i = 1$ until n (where $i = n$ means $i = 0$)
  - We keep a list of possible states for the pair $(x[i-1], x[i])$ (just the chosen possibility when i=1)
  - The pair $(x[i+1], x[i])$ is possible iff $f(x[i-1], x[i], x[i+1]) = s[i]$
- Check if the initial tuple is in our list after the last step.
- If the initial tuple is in the list we found a previous assignment in $\mathcal{O}(n \cdot m^3)$

I – Index Case

Problem

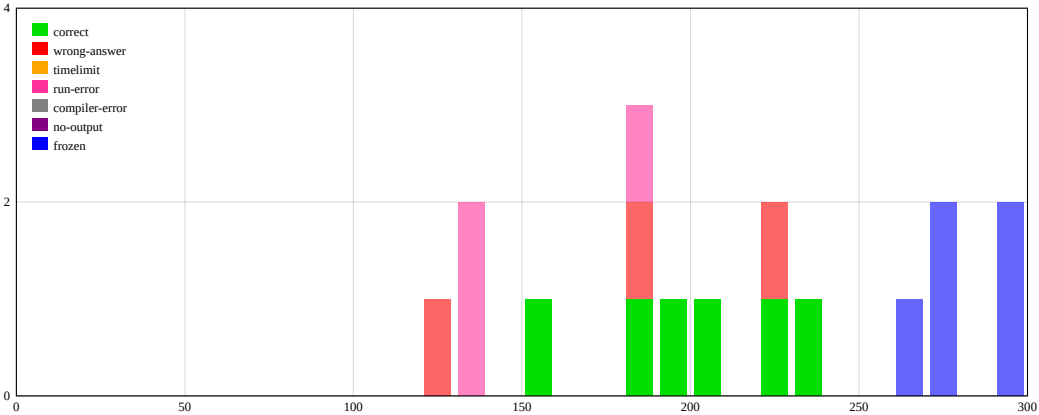
Given a cyclic *cellular automaton* with state s and transition function f, check if a state x exists with $f(x) = s$.

Solution

- First we choose a pair $(x[0], x[1])$
- Starting from $i = 1$ until n (where $i = n$ means $i = 0$)
  - We keep a list of possible states for the pair $(x[i-1], x[i])$ (just the chosen possibility when i=1)
  - The pair $(x[i+1], x[i])$ is possible iff $f(x[i-1], x[i], x[i+1]) = s[i]$
- Check if the initial tuple is in our list after the last step.

- If the initial tuple is in the list we found a previous assignment in $\mathcal{O}(n \cdot m^3)$
- Since we need to consider all initial pairs, the total runtime is $\mathcal{O}(n \cdot m^5)$

N – Natural Navigation



N – Natural Navigation

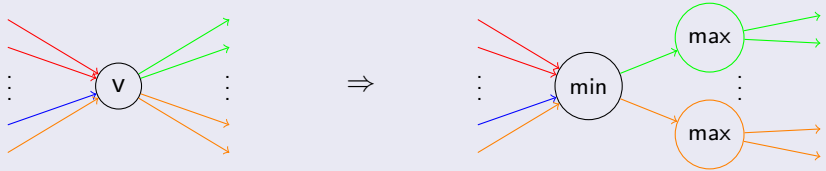
Problem

Given is a directed graph where each edge has one or multiple colours. At any vertex, instead of directly choosing an edge to follow, you can only choose a colour and an opponent will then select the worst possible choice among edges of that colour. Compute the length of the shortest path leading from start to target vertex under these conditions or decide that it's impossible to get there.

N – Natural Navigation

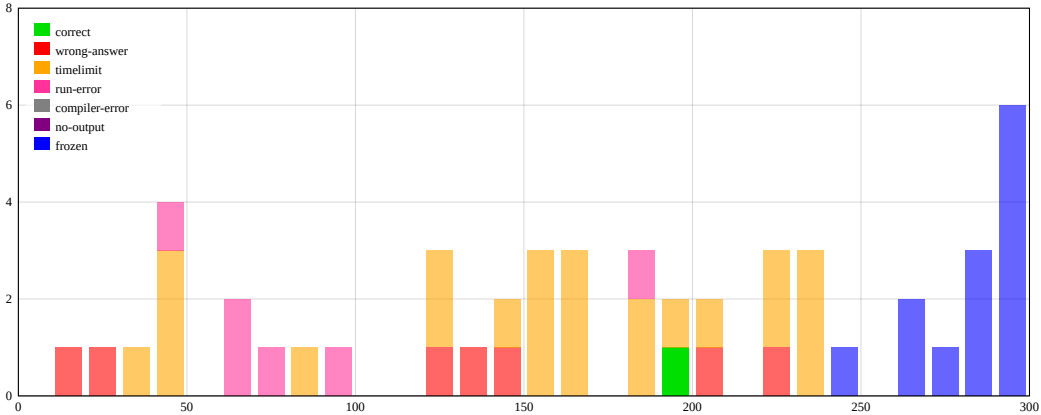
Solution

- Let's split up edges with multiple colors into multiple edges with one colour each.
- Let's split each vertex into its min and max components:



- Solve the problem backwards: What's the distance to the target? Initially 0 for the target and ∞ for all other vertices.
- Update a min vertex when at least one successor has finite distance and a max vertex when all successors have finite distance.
- Always update a vertex with smallest distance $\Rightarrow$ every vertex updated only once.
- Use a priority queue (like in Dijkstra's Algorithm) for runtime $\mathcal{O}(m \log m)$.

D – Decrypting Zodiac



D – Decrypting Zodiac

Problem

Given an encrypted message and a its assumed decryption, at least how many letters where encrypted wrong?

The encryption consists of two steps:

- Caesar shift all letters by the same amount.
- Split the text at any point and switch the two parts.

D – Decrypting Zodiac

Problem

Given an encrypted message and a its assumed decryption, at least how many letters where encrypted wrong?
The encryption consists of two steps:

- Caesar shift all letters by the same amount.
- Split the text at any point and switch the two parts.

Solution

- Note that the second operation is actually a shift/rotation of the text.

$\Rightarrow$ If we encode all chars unary both operations behave similar.
a is encoded as 1000..., b as 0100... and so on

- A Caesar shift by 26 now corresponds to a Split before the last character.

D – Decrypting Zodiac

Problem

Given an encrypted message and a its assumed decryption, at least how many letters where encrypted wrong?
The encryption consists of two steps:

- Caesar shift all letters by the same amount.
- Split the text at any point and switch the two parts.

Solution

- Note that the second operation is actually a shift/rotation of the text.

$\Rightarrow$ If we encode all chars unary both operations behave similar.
a is encoded as 1000..., b as 0100... and so on

- A Caesar shift by 26 now corresponds to a Split before the last character.
- However, a Caesar shift of something less can break everything i.e. za shifted by 1 result in a zero char and one char with two ones.

D – Decrypting Zodiac

Solution

- But we can fix the Caesar shift:
  - In the first string add an empty character after each character
    i.e. "abc" becomes "a b c "
  - In the second string double each character
    i.e. "abc" becomes "aabbcc"
- A shift by $a \cdot 2 \cdot 26 + b$ now corresponds to a Caesar shift by b and a rotation by a in the unary encoding.

D – Decrypting Zodiac

Solution

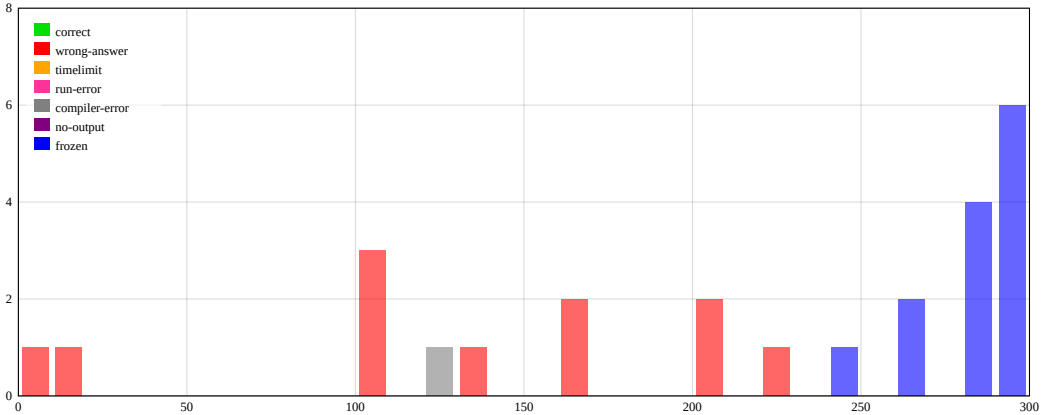
- But we can fix the Caesar shift:
  - In the first string add an empty character after each character
    i.e. "abc" becomes "a b c "
  - In the second string double each character
    i.e. "abc" becomes "aabbcc"
- A shift by $a \cdot 2 \cdot 26 + b$ now corresponds to a Caesar shift by b and a rotation by a in the unary encoding.
- Now we only search a rotation which maximizes the number of bits in the bit-wise and of those two strings. This is also called the cross-correlation of the two strings.

D – Decrypting Zodiac

Solution

- The cross correlation can be computed with the *Fast Fourier transformation* for every rotation by reversing one of the strings.
- Note that we are only interested in rotations of $a \cdot 2 \cdot 26 + b$ with $0 \leq b < 26$.
- The total runtime then is $\mathcal{O}(4 \cdot 26 * n * \log(n))$

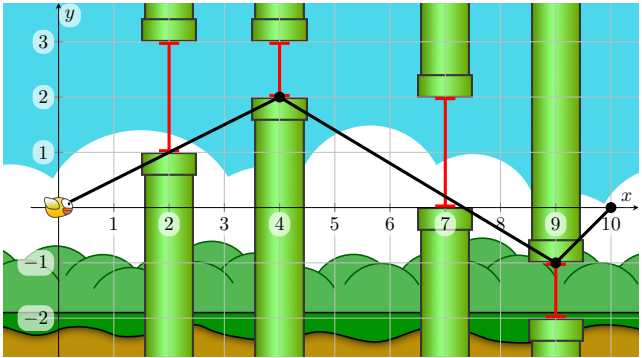
F – Flappy Bird



F – Flappy Bird

Problem

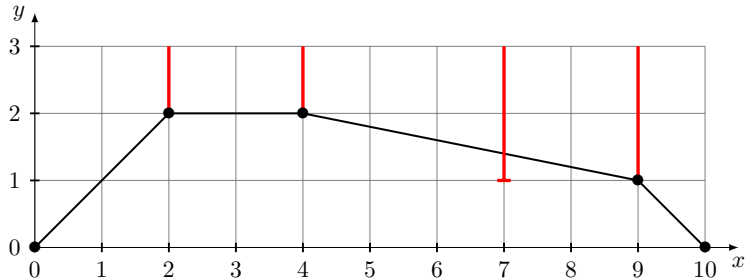
Given two points s and t in the plane and n intervals on the y-axis. Find the shortest polyline connecting s and t, which passes through all intervals in increasing order of their x coordinates.



F – Flappy Bird

This problem is related to convex hulls:

- For intervals of the form $(x_i, y_{i,1}, \infty)$ the answer are the points on the convex hull of $\{s, t, (x_i, y_{i,1})\}$:



- Symmetrical for intervals of the form $(x_i, -\infty, y_{i,2})$

F – Flappy Bird

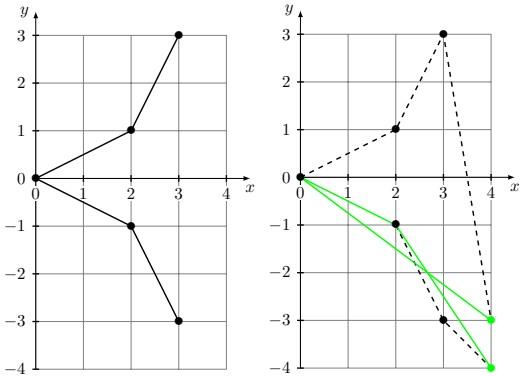
Idea

Compute upper and lower convex hull simultaneously in "Graham-scan style" and merge appropriately.

F – Flappy Bird

Idea

Compute upper and lower convex hull simultaneously in “Graham-scan style” and merge appropriately.



- Before adding the new interval (green points), the hulls only share their leftmost point
- The suffix violating convexity is popped (dashed-lines)
- Updated hulls (green lines) intersect

F – Flappy Bird

Observation

If the convex hulls intersect after adding an interval, exactly one of them became a line.

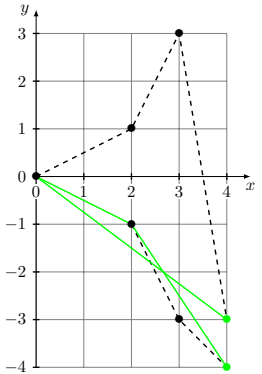
F – Flappy Bird

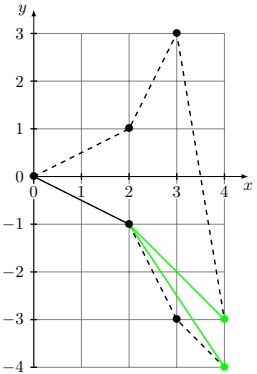
Observation

If the convex hulls intersect after adding an interval, exactly one of them became a line.

Merging hulls

Move leftmost point further right, until top hull is above bottom hull (cross product). The points removed are part of the answer.





F – Flappy Bird

Observation

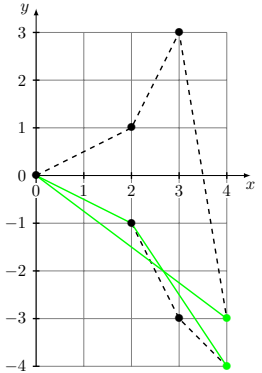
If the convex hulls intersect after adding an interval, exactly one of them became a line.

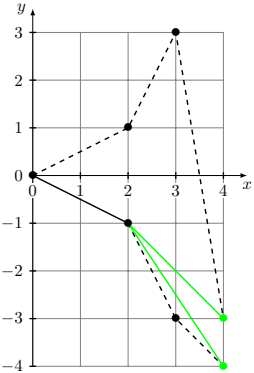
Merging hulls

Move leftmost point further right, until top hull is above bottom hull (cross product). The points removed are part of the answer.

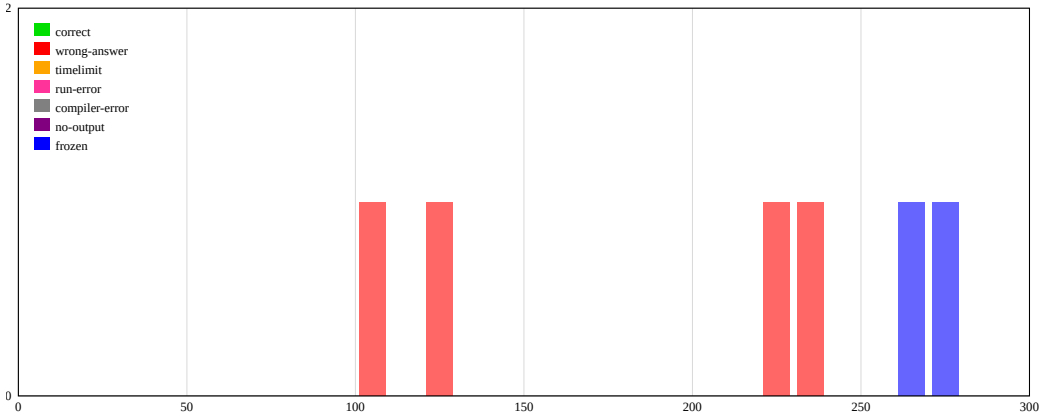
Runtime

Representing hulls using deques yields $\mathcal{O}(n)$ runtime.





J – Joined Sessions



J – Joined Sessions

Problem

Given n intervals. We can merge two intervals if they intersect. A set of intervals S is dominating if every interval is in S or intersects any interval in S. Find the minimum number of unions of intervals that are necessary to reduce the size of a minimum dominating set by at least one.

J – Joined Sessions

Observations

- 3 unions are always sufficient.
- For an interval I_j let last(j) be the index of the interval with the highest right endpoint ending before I_j.
- For an interval I_j let cont(j) be the index of the interval with the lowest left endpoint that intersects I_j.
- It is never necessary to merge an interval I_j with any other interval than cont(j).

J – Joined Sessions

Solution

- Sort the intervals in increasing order of their right endpoints.
- Use dynamic programming, where $\mathrm{dp}(j, s)$ is the minimum number of intervals needed to cover intervals $I_1, \ldots, I_j$ using s unions.

$$\mathrm{dp}(j, s) = \min\left(\mathrm{dp}(\mathrm{last}(j), s) + 1, \min\{\mathrm{dp}(\mathrm{cont}^t(j), s - t) \mid 1 \leq t \leq s\}\right)$$

- Take care of edge cases where there are multiple components or minimum dominating sets of size 1.
- Runtime: $\mathcal{O}(n \log n)$.
- Note: There is a greedy approach with the same runtime.

Weiteres Programm

- Jetzt: Auflösung des Scoreboards und Siegerehrung
- Anschließend Treffen im WorkAdventure
- Extended Contest mit den GCPC-Aufgaben (bald) unter

  `https://domjudge.cs.fau.de/`

Danke für die Teilnahme!