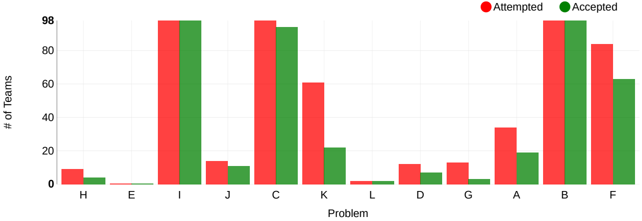
Problem Analysis Session

SWERC judges

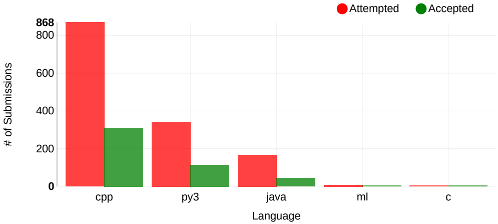
January 26, 2020

Statistics

Number of submissions: about 1400

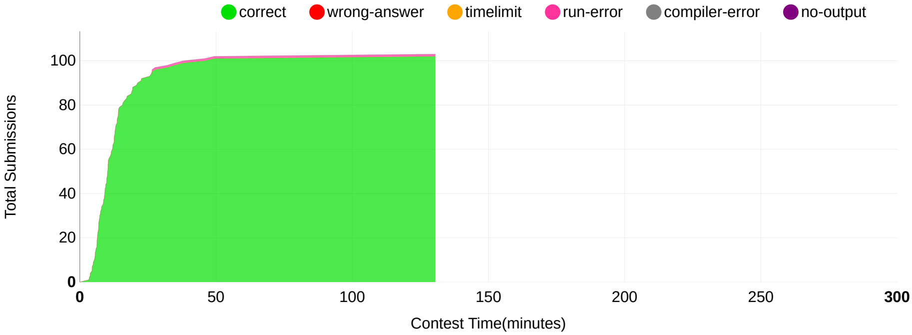


Number of clarification requests: 35 (28 answered “No comment.”)



I – Rats

Solved by all the teams before freeze.
First solved after 3 min by
UnivRennes ISTIC.



I – Rats

This was the easiest problem of the contest.

Problem

Use mark-recapture to estimate the size of an animal population.

Solution

Given

- n_1: number of animals captured and marked on the first day
- n_2: number of animals captured on the second day
- n_{12}: number of animals recaptured (and thus already marked)

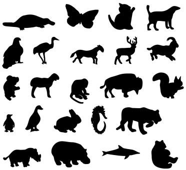
Use the Chapman estimator formula directly:

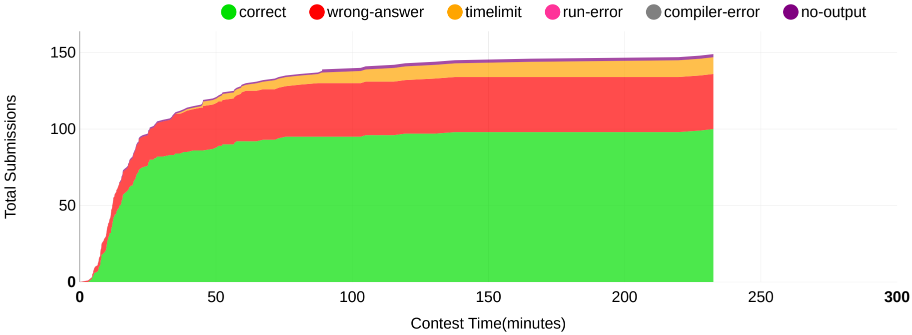
$$\text{population size} = \left\lfloor \frac{(n_1 + 1)(n_2 + 1)}{n_{12} + 1} - 1 \right\rfloor$$

No need for floats.

B – Biodiversity

Solved by all the teams before freeze.
First solved after 3 min by **Télécommander**.





B – Biodiversity

Problem (easy)

Computing the majority (more than half of the total) of N strings.

Solution (linear time and space)

Use a standard **hash table** to compute the number of occurences of each string and look for one that appears more than $N/2$ times.

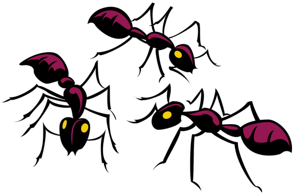
Alternate Solution (linear time and space)

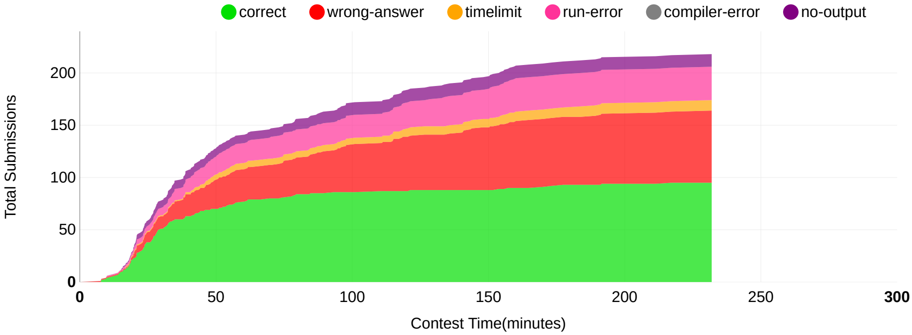
Use Boyer-Moore algorithm to find a candidate and check if the candidate actually appears more than $N/2$ times:

$count \leftarrow 0$
for each string S **do**
  if $count = 0$ **then** $candidate \leftarrow S$
  if $S = candidate$ **then** $count \leftarrow count + 1$
  else $count \leftarrow count - 1$

C – Ants

Solved by 94 teams before freeze.
First solved after 7 min by
Rubber Duck Forces.





C – Ants

This was an easy problem.

Problem: Minimum Excluded a.k.a. mex

Find the smallest *natural* number out of $\{X_1, X_2, \ldots, X_N\}$.

Straightforward solution

Store all the X_i in a set (e.g. a hash table), then linearly check for 0,1,...

A better solution

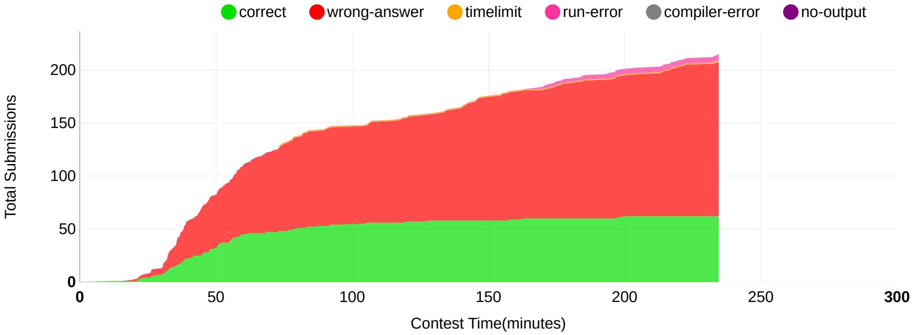
- The answer necessarily belongs to $\{0, 1, \ldots, N\}$.
- So we can use an array and ignore values out of that interval.

A more challenging variant

Do it in place.

F – Icebergs

Solved by 63 teams before freeze.
First solved after 15 min by **UPC-1**.



F – Icebergs

Problem

Given N polygons, compute their total area.

F – Icebergs

Problem

Given N polygons, compute their total area.

Remark

Polygons can be treated separately.

F – Icebergs

Problem

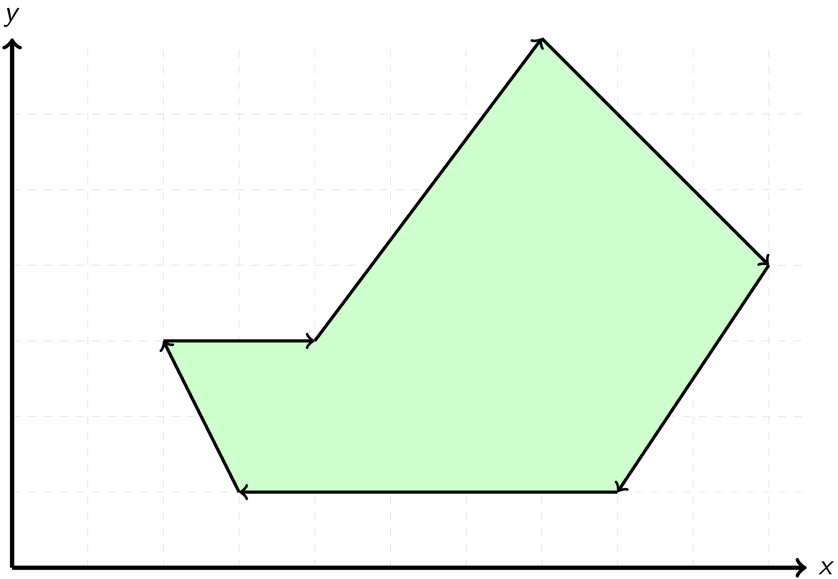
Given N polygons, compute their total area.

Remark

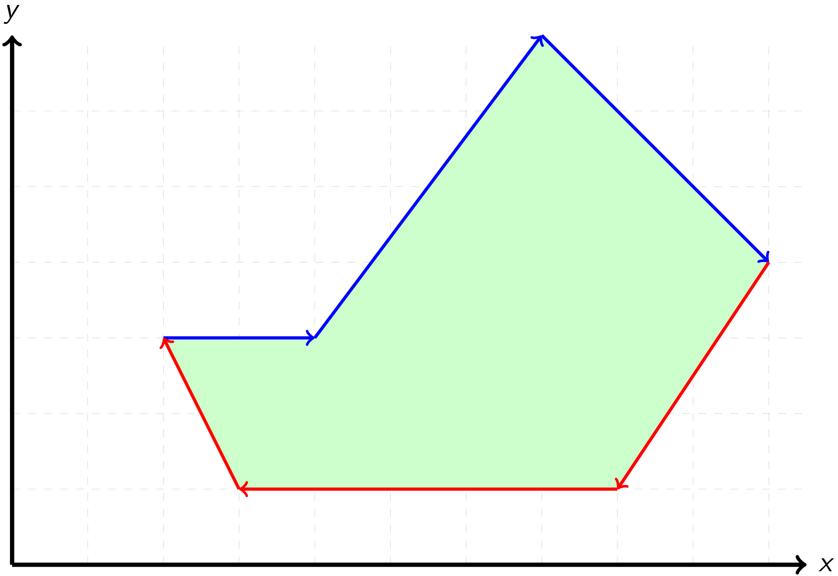
Polygons can be treated separately.

$\Rightarrow$ How to compute the area of a polygon?

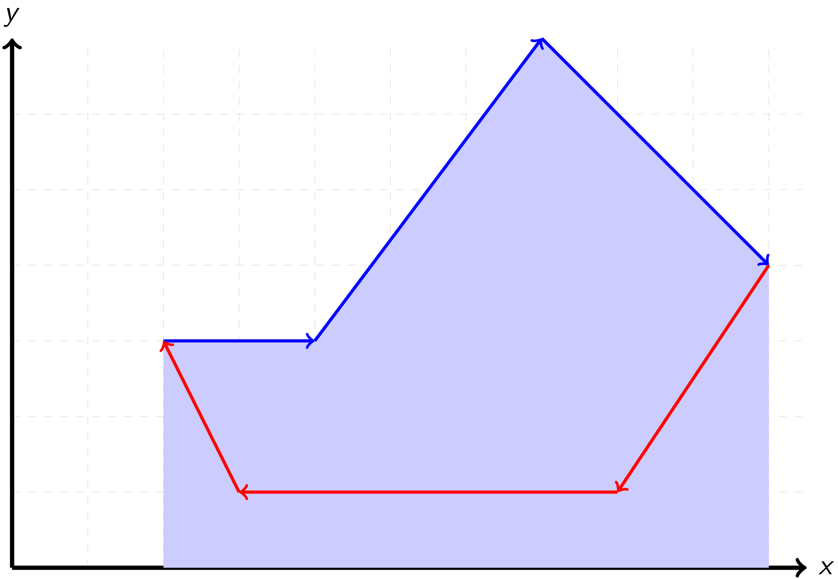
F – Icebergs



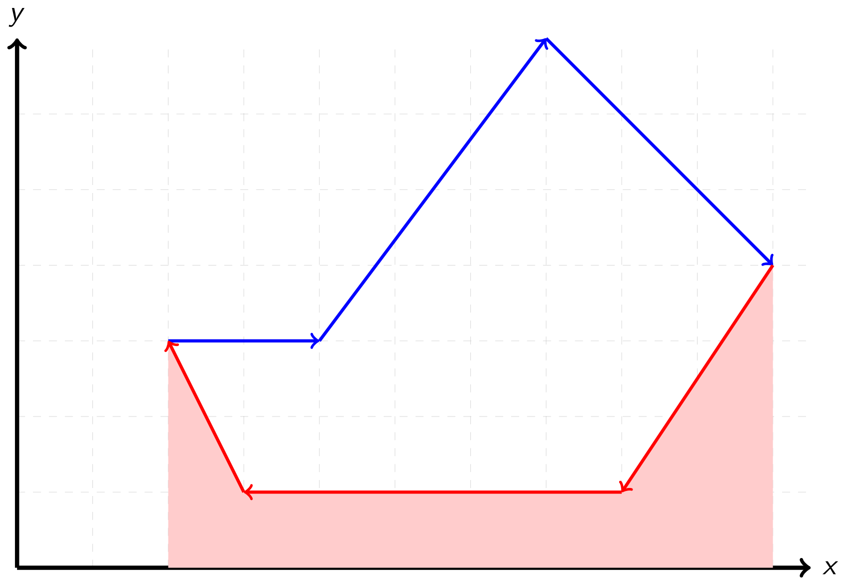
F – Icebergs



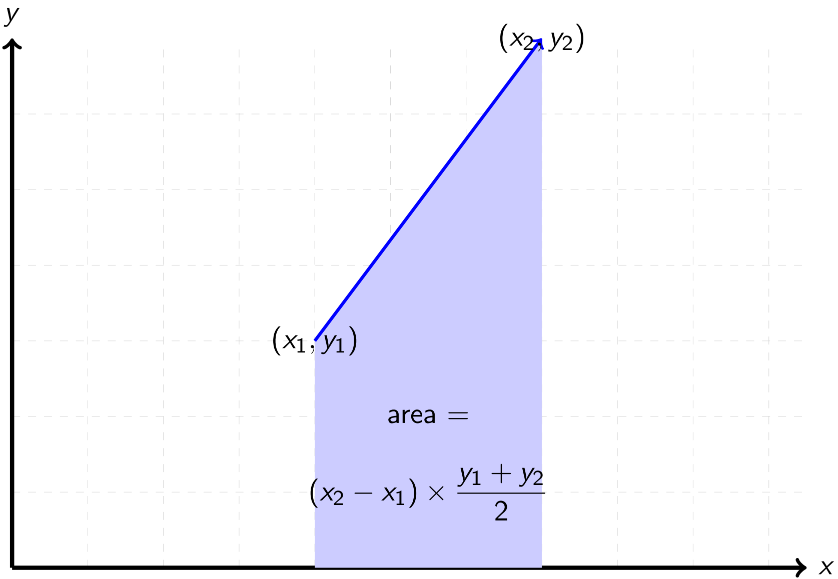
F – Icebergs



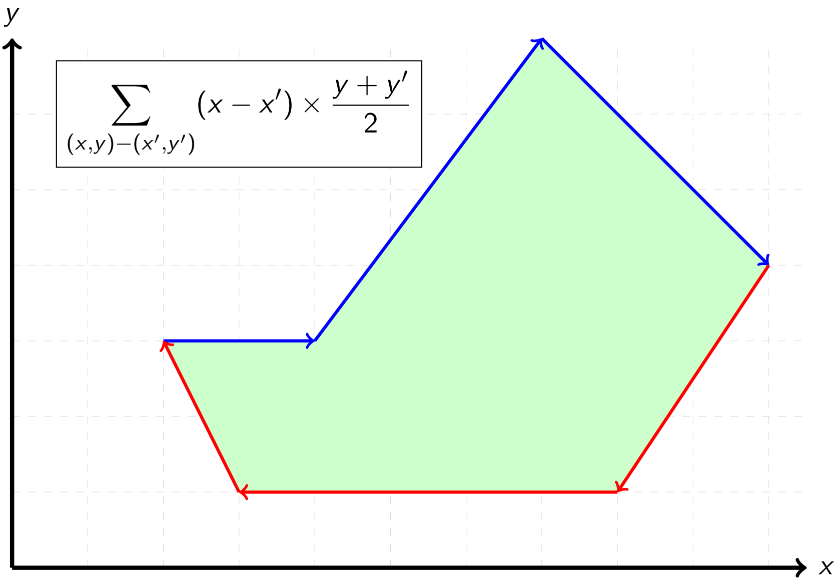
F – Icebergs



F – Icebergs

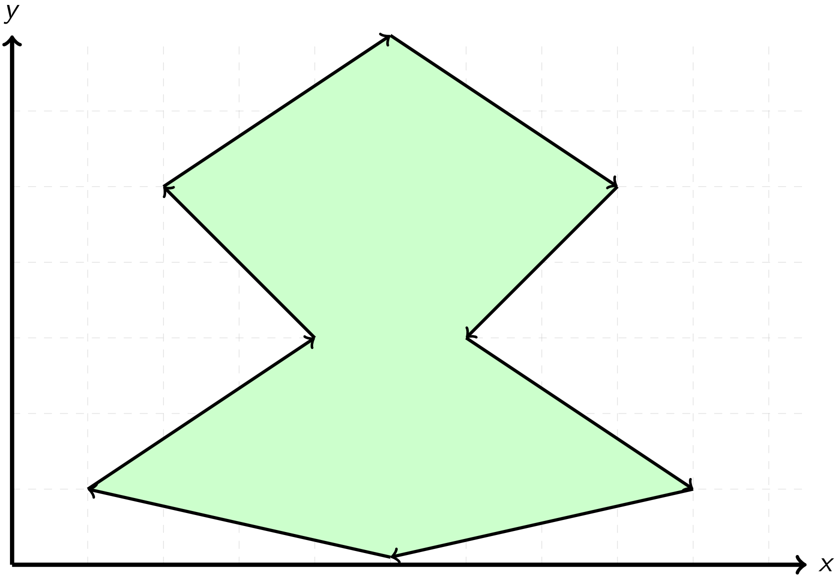


F – Icebergs

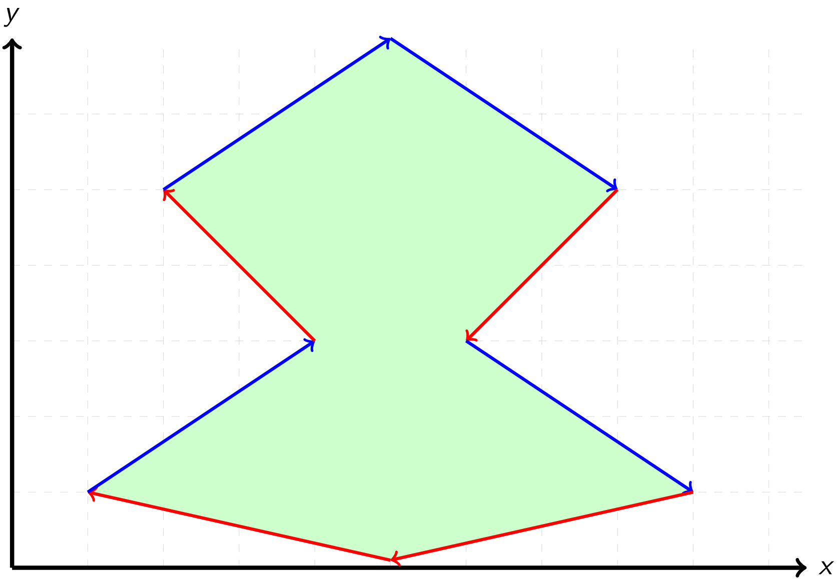


$$\sum_{(x,y)-(x',y')} (x - x') \times \frac{y + y'}{2}$$

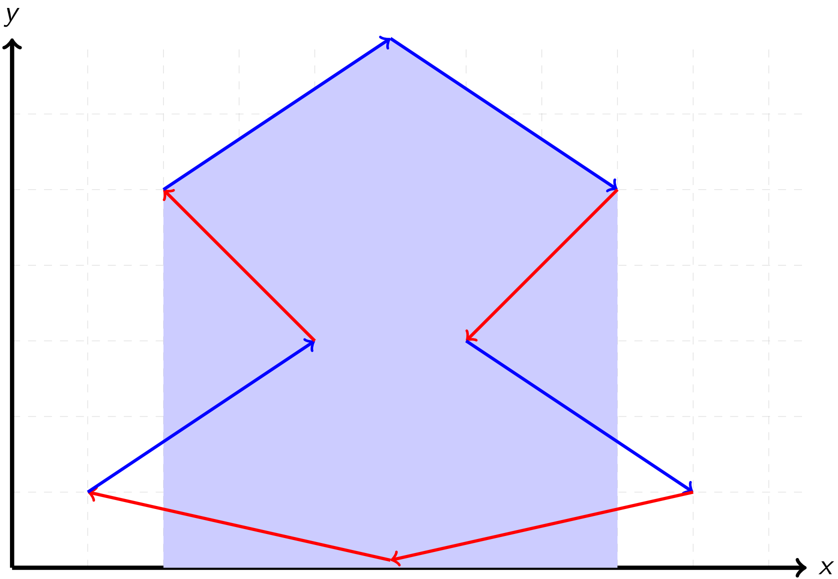
F – Icebergs



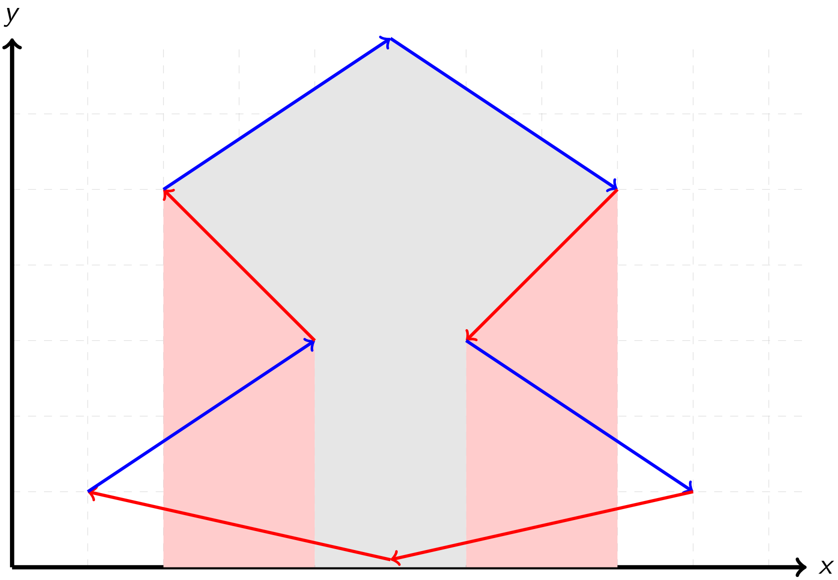
F – Icebergs



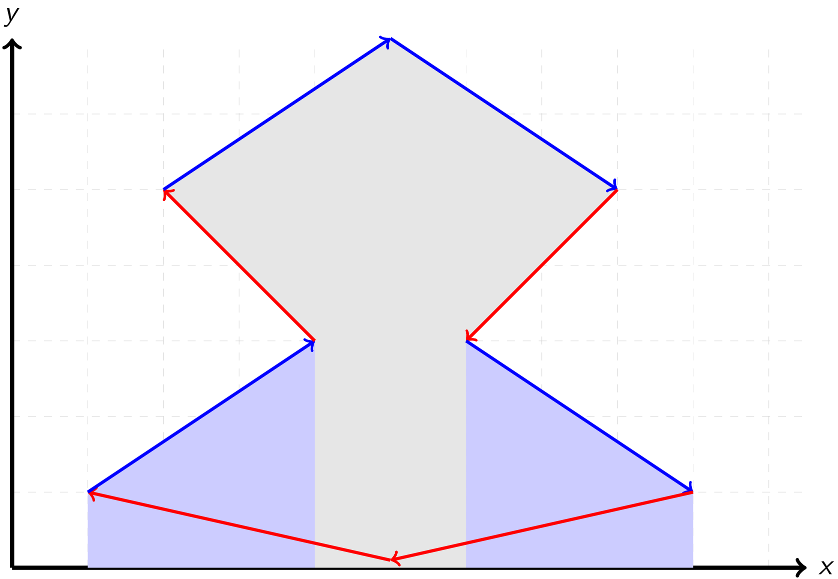
F – Icebergs



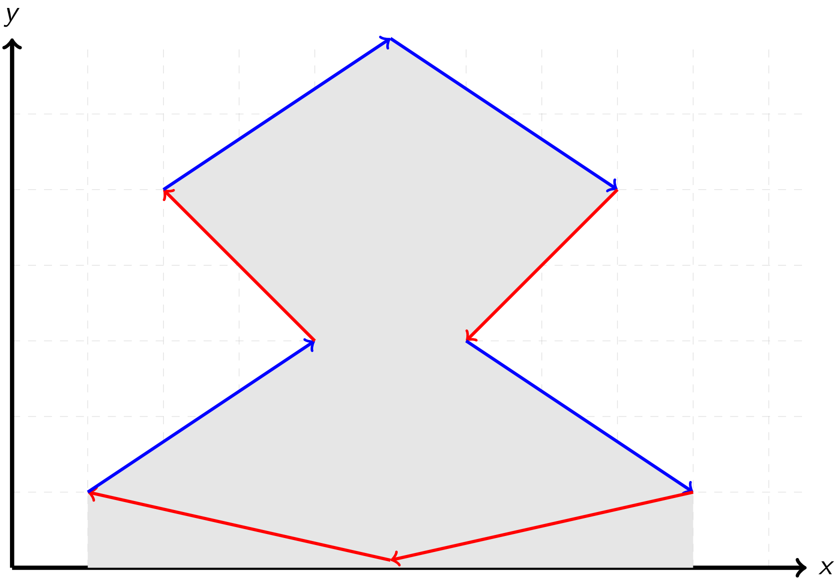
F – Icebergs



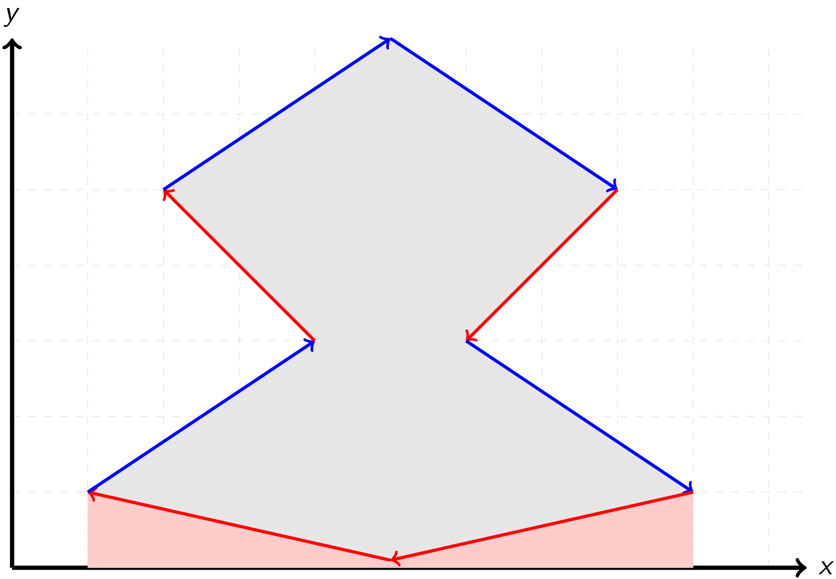
F – Icebergs



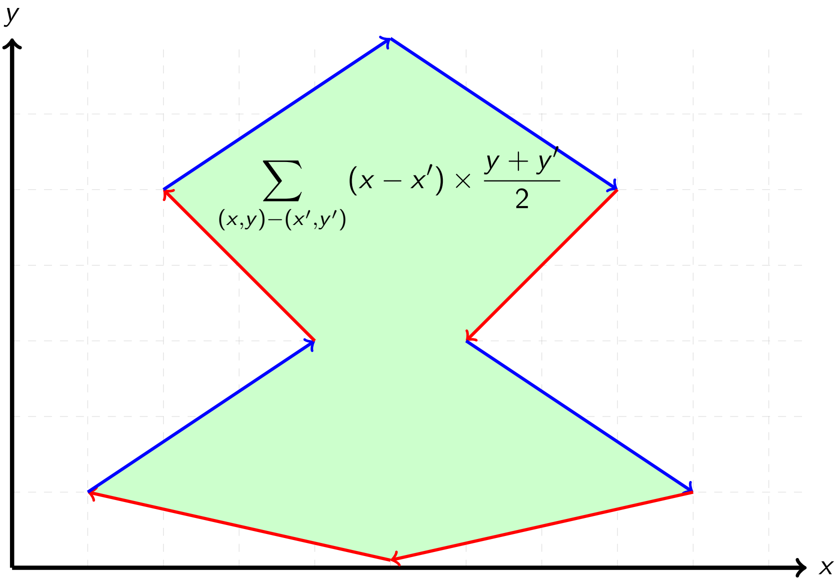
F – Icebergs



F – Icebergs

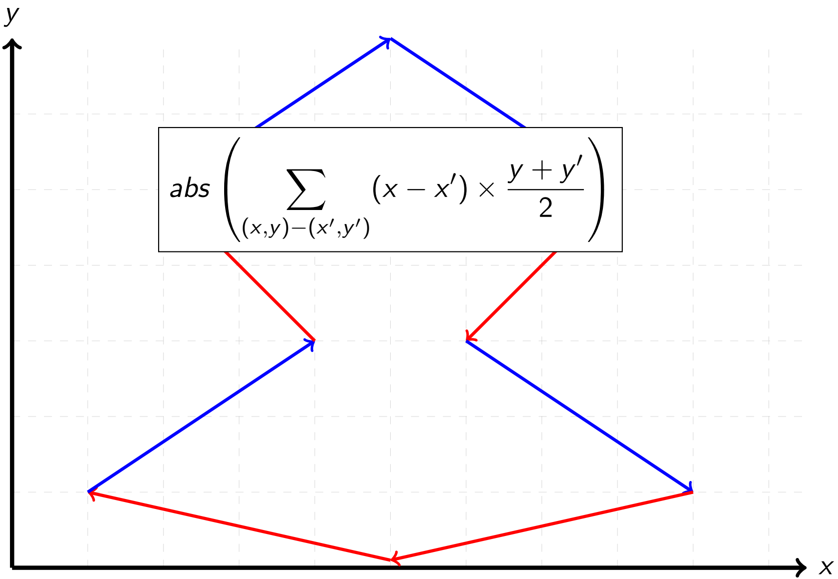


F – Icebergs



$$\sum_{(x,y)-(x',y')} (x - x') \times \frac{y + y'}{2}$$

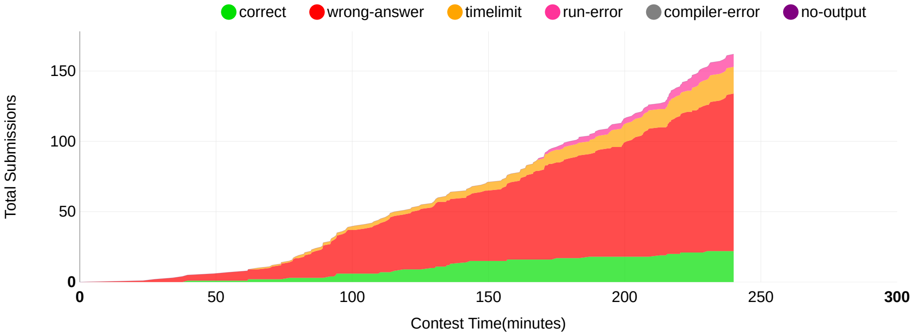
F – Icebergs



$$abs\left(\sum_{(x,y)-(x',y')}(x-x')\times\frac{y+y'}{2}\right)$$

K – Bird Watching

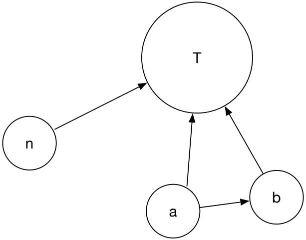
Solved by 22 teams before freeze.
First solved after 39 min by **UPC-1**.



K – Bird Watching

Problem

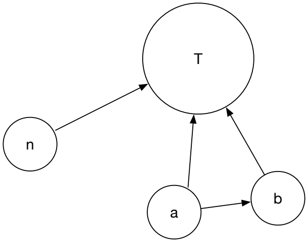
Given a vertex T in a directed graph $\mathcal{P}$, find all nodes n such that the edge (n, T) is the only path from n to T.



K – Bird Watching

Problem

Given a vertex T in a directed graph $\mathcal{P}$, find all nodes n such that the edge (n, T) is the only path from n to T.



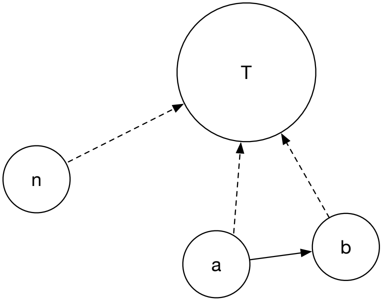
Naive approach

Remove (n, T) and check whether you can still reach T.
This requires $|V|$ DFSs, i.e., $|V| \times |E| \approx 10^{10}$ operations.
$\Rightarrow$ How do we cut the search?

K – Bird Watching

Auxiliary graph

$\mathcal{P}^*$: Remove all edges leading to T

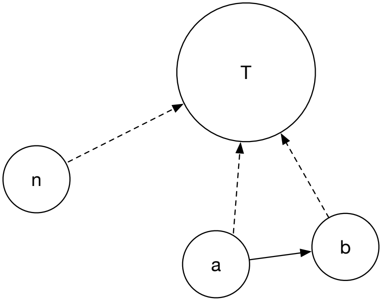


K – Bird Watching

Auxiliary graph

$\mathcal{P}^*$: Remove all edges leading to T

n is a solution when there is no other node n' where the edge $n' \to T$ is in $\mathcal{P}$ and there is a path from n to n' in $\mathcal{P}^*$.

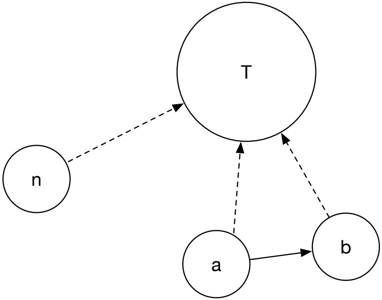


n is a solution, a is not ($b = n'$)

K – Bird Watching

Simplified algorithm

For each n, find some n' satisfying the previous requirements and stop the search to cut branches.



K – Bird Watching

Simplified algorithm

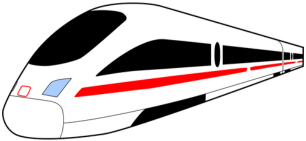
Call annotate(r, r) for each r predecessor of T:

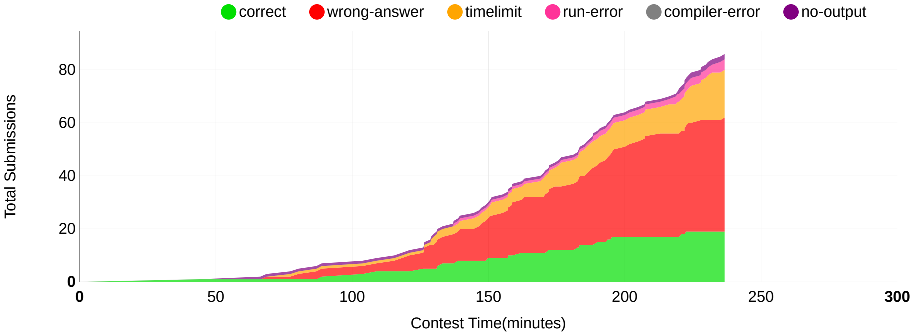
- goal(n) is a set of predecessors of T that are accessible from n in $\mathcal{P}^*$ (with at most 2 elements)
- a predecessor n of T is a solution iff $|\text{goal}(n)| = 1$ (contains only n).

```
annotate(n, r):
  if r ∈ goal(n): stop
  if |goal(n)| ⩾ 2: stop
  goal(n) ← goal(n) ∪ {r}
  for each (u, n) ∈ P*: annotate(u, r)
```

A – Environment-Friendly Travel

Solved by 19 teams before freeze.
First solved after 43 min by **UNIBOis**.

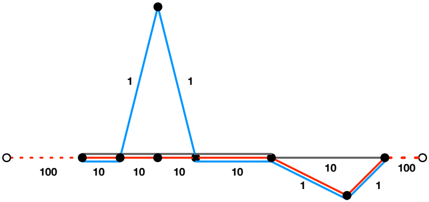




A – Environment-Friendly Travel

Problem

Given two distances d_1 (CO_2-cost), d_2 (Euclidean distance) on a graph G, find the smallest $d_1(s,t)$ such that $d_2(s,t) \leqslant B$.



E.g., for a budget of $B = 15$:

1. The **shortest path** (distance) costs too much CO_2.
2. The **cheapest path** is too long.
3. The best one is the **red** path.

A – Environment-Friendly Travel

Problem

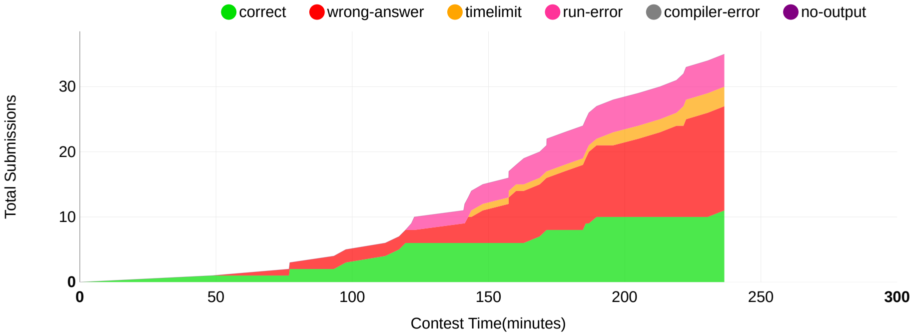
Given two distances d_1 (CO_2), d_2 (Euclidean distance) on a graph G, find the smallest $d_1(s, t)$ such that $d_2(s, t) \leqslant B$.

Solution

Run a shortest-path algorithm (Dijkstra) on the cost graph (d_1), keep only the paths for which $d_1 \leqslant B$.

J – Counting Trees

Solved by 11 teams before freeze.
First solved after 48 min by **ENS Ulm 1**.



J – Counting Trees

Problem: Cartesian trees

Count the number of integer-labelled binary trees which:

- have the min-heap property, and
- have a given integer sequence as their in-order traversal.

Basic DP solution (too slow)

How many trees for a given sub-sequence? Complexity: $\mathcal{O}(n^3)$.

J – Counting Trees

Choosing one of these trees:

$$2, 3, 1, 2, 2, 1, 1, 3, 2, 3$$

J – Counting Trees

Choosing one of these trees:

1. Locate the occurrences of the minimum of the sequence

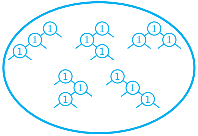
$$2, 3, 1, 2, 2, 1, 1, 3, 2, 3$$

J – Counting Trees

Choosing one of these trees:

1. Locate the occurrences of the minimum of the sequence
2. Choose an arrangement of these nodes at the top of the tree
   - Number of choices: Catalan number $C_n = \frac{1}{n+1}\binom{2n}{n}$

$2, 3, 1, 2, 2, 1, 1, 3, 2, 3$

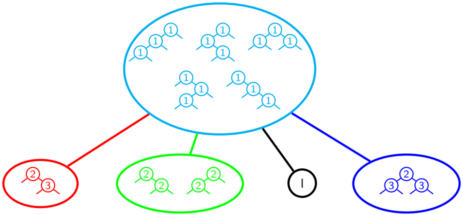


J – Counting Trees

Choosing one of these trees:

1. Locate the occurrences of the minimum of the sequence
2. Choose an arrangement of these nodes at the top of the tree
   - Number of choices: Catalan number $C_n = \frac{1}{n+1}\binom{2n}{n}$
3. Choose the sub-trees recursively, for each of the delimited sub-sequences.

2, 3, 1, 2, 2, 1, 1, 3, 2, 3

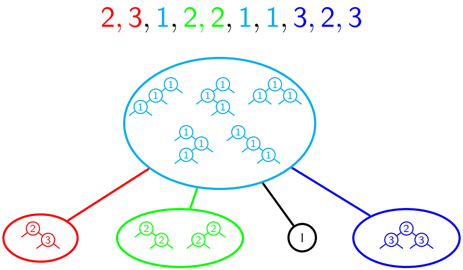


J – Counting Trees

Choosing one of these trees:

1. Locate the occurrences of the minimum of the sequence
2. Choose an arrangement of these nodes at the top of the tree
   - Number of choices: Catalan number $C_n = \frac{1}{n+1}\binom{2n}{n}$
3. Choose the sub-trees recursively, for each of the delimited sub-sequences.

Total complexity: $\mathcal{O}(n^2)$, or $\mathcal{O}(n \log n)$ with a min-range data structure.



The result is a product of Catalan numbers.

Each factor C_n corresponds to a group of n elements of the sequence which:

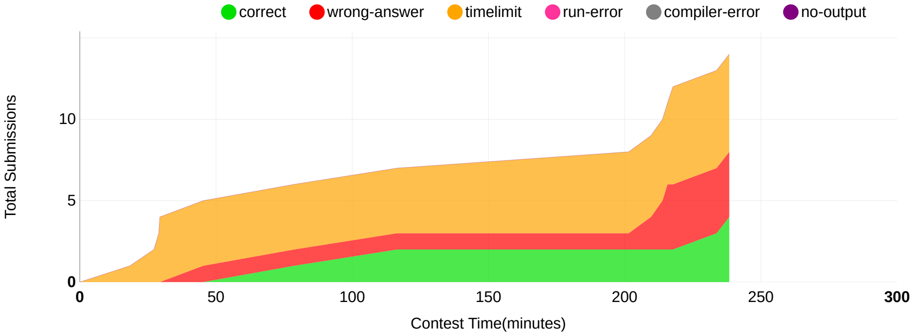
- have the same value,
- is not separated by a smaller element.

We can compute these groups using a stack in one pass on the sequence.

$\Rightarrow \mathcal{O}(n)$ algorithm

$\Rightarrow$ All included: 15 lines of simple Python code!

Solved by 4 teams before freeze.
First solved after 78 min by **LaStatale Blue**.



H – Pseudo-Random Number Generator

Problem

A pseudo-random number generator for 40-bit unsigned integers is defined as the iteration of a function f, that is,

$$\begin{aligned} S_0 &= \text{some given value}, \\ S_{i+1} &= f(S_i). \end{aligned}$$

Find the number of even values in the sequence $S_0, S_1, \ldots, S_{N-1}$.

Limits

The number N can be large (up to 2^{63}) so we cannot simply compute all the values.

H – Pseudo-Random Number Generator

Analysis

Since there are finitely-many values, the sequence S eventually cycles after a certain point: there exist $\mathit{period} \geq 1$ and $\mathit{start} \geq 0$ such that

$$S_{i+\mathit{period}} = S_i \quad \text{for } i \geq \mathit{start}.$$

Idea

Before submission,

- find *period* and *start*;
- pre-compute the number of even values for
  - the whole initial sequence $S_0, S_1, \ldots, S_{\mathit{start}-1}$,
  - the whole cycle $S_{\mathit{start}}, S_{\mathit{start}+1}, \ldots, S_{\mathit{start}+\mathit{period}-1}$,
  - blocks of consecutive S_i (e.g. 1000 blocks in total).

Submit a code that tests whether $N < \mathit{start}$ or $N = \mathit{start} + q \cdot \mathit{period} + r$ with $0 \leq r < \mathit{period}$ and uses the pre-computed values.

H – Pseudo-Random Number Generator

Cycle detection

We are left with the problem of finding *period* and *start*.
Storing all S_i until we find the cycle requires too much memory.

Solution

Use Floyd's *tortoise and hare* algorithm:

$$t,\ h \leftarrow 0, 1$$
$$\textbf{while } S_t \neq S_h \textbf{ do } t,\ h \leftarrow t+1,\ h+2$$
$$i \leftarrow 0$$
$$\textbf{while } S_i \neq S_{t+i} \textbf{ do } i \leftarrow i+1$$

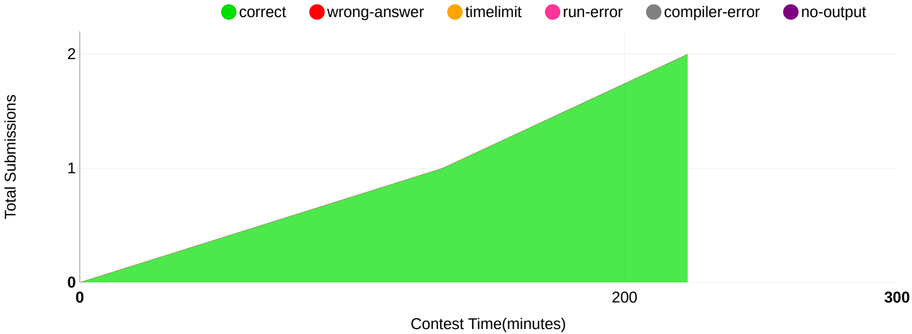
[See *The Art of Computer Programming*, volume 2, page 7, exercise 6.]

Efficiency

Precomputation: $\mathcal{O}(start + period)$.
(In our case, $period = 182\,129\,209$ and $start = 350\,125\,310$.)
Submitted solution is $\mathcal{O}(1)$.

L – River Game

Solved by 2 teams before freeze.
First solved after 133 min by **EP Chopper**.



L – River Game

Problem

Two players take turns to place cameras on a $N \times N$ grid with firm ground, wet zone and protected zone squares. Rivers are connected components of wet squares.

Rules:

- Cameras must be on firm ground, adjacent to a river.
- No two cameras on same square.
- No two cameras adjacent to the same river can be adjacent.

River properties:

- Contain at most $2N$ squares.
- Any two squares from two different rivers are at least 3 squares apart.

Who will win the game (assuming optimal play)?

Limits: $N \leq 10$

L – River Game

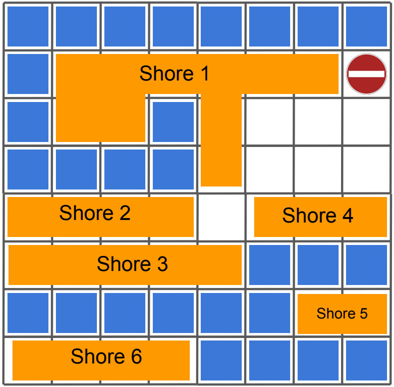
Brute force solution

- State is the $N \times N$ grid with already placed cameras marked.
- Complexity $\geq \mathcal{O}(2^{N \times N})$. Too slow.

L – River Game

Faster solution: Key idea

- **Shore**: connected component of firm ground squares adjacent to a given river.
- Cameras on one shore don't affect cameras on other shores!



L – River Game

Faster solution

- Decompose the game into K independent games (K = number of shores, $\leq N^2$ and in practice much less).
- Compute the Grundy number G_i of each shore. Computed in $\mathcal{O}(S \times 2^S)$ where S = maximum shore size.
- Position is losing iff $G_0 \oplus \ldots \oplus G_K = 0$.
- $S \leq 3N + o(3N)$. For $N = 10$ bound is tighter: $S \leq 20$.
- For $N = 10$, takes less than 100×2^{20} operations.

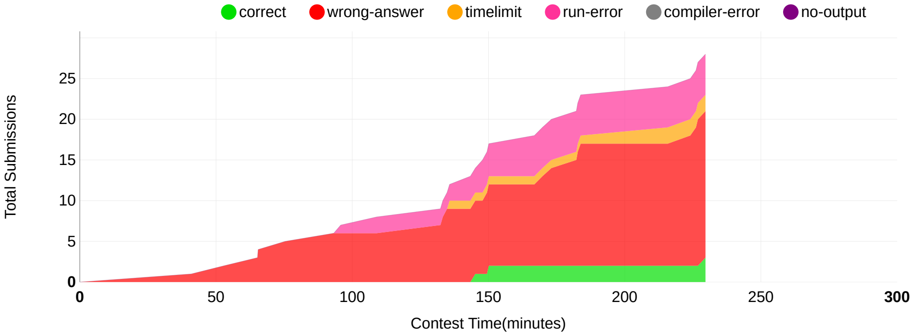
L – River Game

Grundy Numbers computation

```
def GrundyNumber(state):
    next_states = list of possible next states after
                  adding a camera
    next_grundy = set()
    for s in next_states:
        next_grundy.add(GrundyNumber(s))
    # Compute smallest non-negative integer not in
    # next_grundy (problem Ants!).
    res = 0
    while res in next_grundy: res += 1
    return res
```

G – Swapping Places

Solved by 3 teams before freeze.
First solved after 145 min by **UPC-1**.



G – Swapping Places

Problem

Given a word w on an alphabet A and a set $S \subseteq A^2$ of pairs of letters that commute with each other, find the smallest word $\overline{w}$ equivalent to w.

Limits

- A is small: $|A| \leqslant 200$;
- w can be long: $|w| \leqslant 100\,000$.

We can work in time $\mathcal{O}(|A|^2\,|w|)$ but not $\Omega(|w|^2)$.

G – Swapping Places

Idea

Find the letters of $\overline{w}$ one by one, from left to right:

- For each letter λ, find the longest prefix w_λ of w that commutes with λ and does not contain λ.
- The first letter of $\overline{w}$ is the smallest μ such that $w_\mu \mu$ is a prefix of w.

G – Swapping Places

Idea

Find the letters of $\overline{w}$ one by one, from left to right:

- For each letter λ, find the longest prefix w_λ of w that commutes with λ and does not contain λ.
- The first letter of $\overline{w}$ is the smallest μ such that $w_\mu \mu$ is a prefix of w.
- Erase the leftmost occurrence of μ in w and update all prefixes w_λ.

G – Swapping Places

Idea

Find the letters of $\overline{w}$ one by one, from left to right:

- For each letter λ, find the longest prefix w_λ of w that commutes with λ and does not contain λ.
- The first letter of $\overline{w}$ is the smallest μ such that $w_\mu \mu$ is a prefix of w.
- Erase the leftmost occurrence of μ in w and update all prefixes w_λ.

Example: $w = 341231$, with $12 = 21$, $14 = 41$, $23 = 32$ and $24 = 42$

w:	3	4	1	2	3	1
w_1:						
w_2:						
w_3:						
w_4:						

$\overline{w}$:

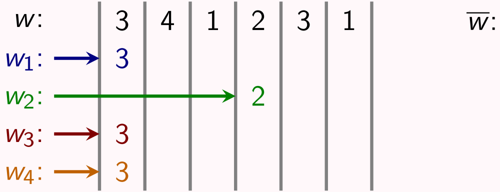
G – Swapping Places

Idea

Find the letters of $\overline{w}$ one by one, from left to right:

- For each letter λ, find the longest prefix w_λ of w that commutes with λ and does not contain λ.
- The first letter of $\overline{w}$ is the smallest μ such that $w_\mu \mu$ is a prefix of w.
- Erase the leftmost occurrence of μ in w and update all prefixes w_λ.

Example: $w = 341231$, with $12 = 21$, $14 = 41$, $23 = 32$ and $24 = 42$



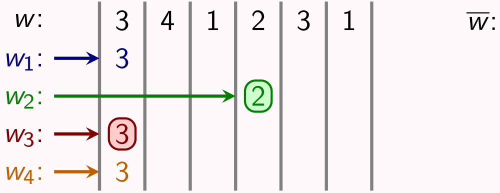
G – Swapping Places

Idea

Find the letters of $\overline{w}$ one by one, from left to right:

- For each letter λ, find the longest prefix w_λ of w that commutes with λ and does not contain λ.
- The first letter of $\overline{w}$ is the smallest μ such that $w_\mu \mu$ is a prefix of w.
- Erase the leftmost occurrence of μ in w and update all prefixes w_λ.

Example: $w = 341231$, with $12 = 21$, $14 = 41$, $23 = 32$ and $24 = 42$

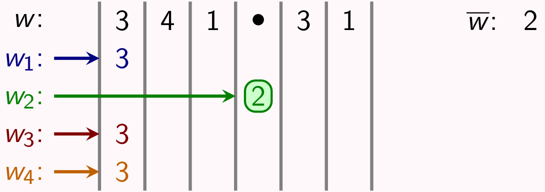


Idea

Find the letters of $\overline{w}$ one by one, from left to right:

- For each letter λ, find the longest prefix w_λ of w that commutes with λ and does not contain λ.
- The first letter of $\overline{w}$ is the smallest μ such that $w_\mu \mu$ is a prefix of w.
- Erase the leftmost occurrence of μ in w and update all prefixes w_λ.

Example: $w = 341231$, with $12 = 21$, $14 = 41$, $23 = 32$ and $24 = 42$



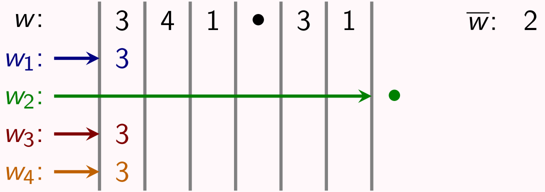
G – Swapping Places

Idea

Find the letters of $\overline{w}$ one by one, from left to right:

- For each letter λ, find the longest prefix w_λ of w that commutes with λ and does not contain λ.
- The first letter of $\overline{w}$ is the smallest μ such that $w_\mu \mu$ is a prefix of w.
- Erase the leftmost occurrence of μ in w and update all prefixes w_λ.

Example: $w = 341231$, with $12 = 21$, $14 = 41$, $23 = 32$ and $24 = 42$



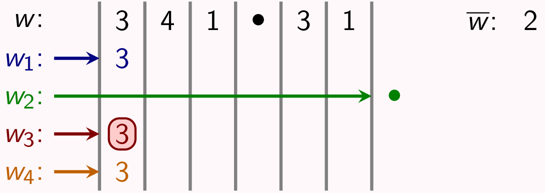
G – Swapping Places

Idea

Find the letters of $\overline{w}$ one by one, from left to right:

- For each letter λ, find the longest prefix w_λ of w that commutes with λ and does not contain λ.
- The first letter of $\overline{w}$ is the smallest μ such that $w_\mu \mu$ is a prefix of w.
- Erase the leftmost occurrence of μ in w and update all prefixes w_λ.

Example: $w = 341231$, with $12 = 21$, $14 = 41$, $23 = 32$ and $24 = 42$



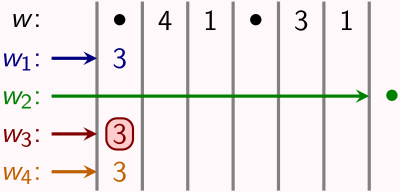
G – Swapping Places

Idea

Find the letters of $\overline{w}$ one by one, from left to right:

- For each letter λ, find the longest prefix w_λ of w that commutes with λ and does not contain λ.
- The first letter of $\overline{w}$ is the smallest μ such that $w_\mu \mu$ is a prefix of w.
- Erase the leftmost occurrence of μ in w and update all prefixes w_λ.

Example: $w = 341231$, with $12 = 21$, $14 = 41$, $23 = 32$ and $24 = 42$



$\overline{w}$: 2 3

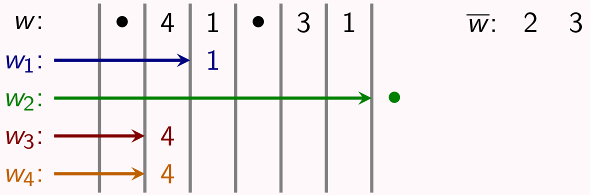
G – Swapping Places

Idea

Find the letters of $\overline{w}$ one by one, from left to right:

- For each letter λ, find the longest prefix w_λ of w that commutes with λ and does not contain λ.
- The first letter of $\overline{w}$ is the smallest μ such that $w_\mu \mu$ is a prefix of w.
- Erase the leftmost occurrence of μ in w and update all prefixes w_λ.

Example: $w = 341231$, with $12 = 21$, $14 = 41$, $23 = 32$ and $24 = 42$



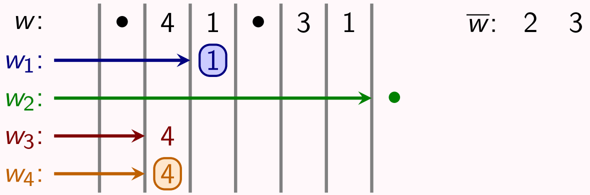
G – Swapping Places

Idea

Find the letters of $\overline{w}$ one by one, from left to right:

- For each letter λ, find the longest prefix w_λ of w that commutes with λ and does not contain λ.
- The first letter of $\overline{w}$ is the smallest μ such that $w_\mu \mu$ is a prefix of w.
- Erase the leftmost occurrence of μ in w and update all prefixes w_λ.

Example: $w = 341231$, with $12 = 21$, $14 = 41$, $23 = 32$ and $24 = 42$



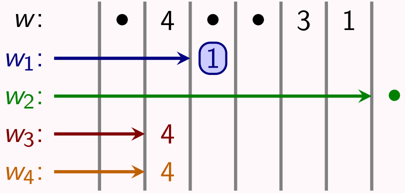
G – Swapping Places

Idea

Find the letters of $\overline{w}$ one by one, from left to right:

- For each letter λ, find the longest prefix w_λ of w that commutes with λ and does not contain λ.
- The first letter of $\overline{w}$ is the smallest μ such that $w_\mu \mu$ is a prefix of w.
- Erase the leftmost occurrence of μ in w and update all prefixes w_λ.

Example: $w = 341231$, with $12 = 21$, $14 = 41$, $23 = 32$ and $24 = 42$



$\overline{w}$: 2 3 1

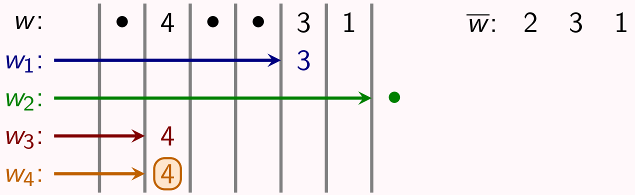
G – Swapping Places

Idea

Find the letters of $\overline{w}$ one by one, from left to right:

- For each letter λ, find the longest prefix w_λ of w that commutes with λ and does not contain λ.
- The first letter of $\overline{w}$ is the smallest μ such that $w_\mu \mu$ is a prefix of w.
- Erase the leftmost occurrence of μ in w and update all prefixes w_λ.

Example: $w = 341231$, with $12 = 21$, $14 = 41$, $23 = 32$ and $24 = 42$



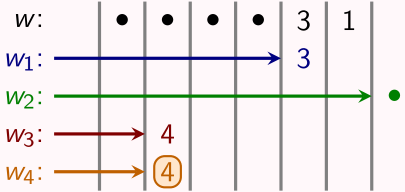
G – Swapping Places

Idea

Find the letters of $\overline{w}$ one by one, from left to right:

- For each letter λ, find the longest prefix w_λ of w that commutes with λ and does not contain λ.
- The first letter of $\overline{w}$ is the smallest μ such that $w_\mu \mu$ is a prefix of w.
- Erase the leftmost occurrence of μ in w and update all prefixes w_λ.

Example: $w = 341231$, with $12 = 21$, $14 = 41$, $23 = 32$ and $24 = 42$



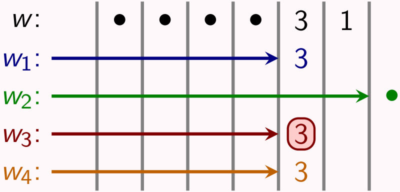
$\overline{w}$: 2 3 1 4

Idea

Find the letters of $\overline{w}$ one by one, from left to right:

- For each letter λ, find the longest prefix w_λ of w that commutes with λ and does not contain λ.
- The first letter of $\overline{w}$ is the smallest μ such that $w_\mu \mu$ is a prefix of w.
- Erase the leftmost occurrence of μ in w and update all prefixes w_λ.

Example: $w = 341231$, with $12 = 21$, $14 = 41$, $23 = 32$ and $24 = 42$



$\overline{w}$: 2 3 1 4

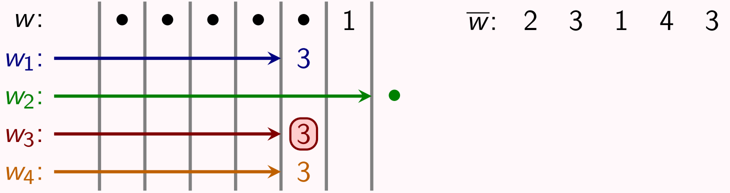
G – Swapping Places

Idea

Find the letters of $\overline{w}$ one by one, from left to right:

- For each letter λ, find the longest prefix w_λ of w that commutes with λ and does not contain λ.
- The first letter of $\overline{w}$ is the smallest μ such that $w_\mu \mu$ is a prefix of w.
- Erase the leftmost occurrence of μ in w and update all prefixes w_λ.

Example: $w = 341231$, with $12 = 21$, $14 = 41$, $23 = 32$ and $24 = 42$



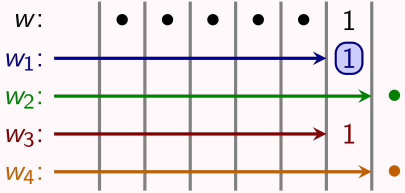
G – Swapping Places

Idea

Find the letters of $\overline{w}$ one by one, from left to right:

- For each letter λ, find the longest prefix w_λ of w that commutes with λ and does not contain λ.
- The first letter of $\overline{w}$ is the smallest μ such that $w_\mu \mu$ is a prefix of w.
- Erase the leftmost occurrence of μ in w and update all prefixes w_λ.

Example: $w = 341231$, with $12 = 21$, $14 = 41$, $23 = 32$ and $24 = 42$



$\overline{w}$: 2 3 1 4 3

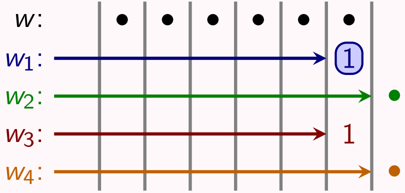
G – Swapping Places

Idea

Find the letters of $\overline{w}$ one by one, from left to right:

- For each letter λ, find the longest prefix w_λ of w that commutes with λ and does not contain λ.
- The first letter of $\overline{w}$ is the smallest μ such that $w_\mu \mu$ is a prefix of w.
- Erase the leftmost occurrence of μ in w and update all prefixes w_λ.

Example: $w = 341231$, with $12 = 21$, $14 = 41$, $23 = 32$ and $24 = 42$



$\overline{w}$: 2 3 1 4 3 1

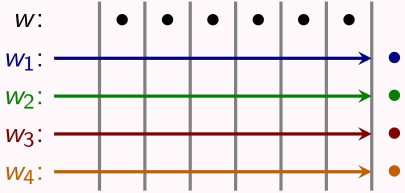
G – Swapping Places

Idea

Find the letters of $\overline{w}$ one by one, from left to right:

- For each letter λ, find the longest prefix w_λ of w that commutes with λ and does not contain λ.
- The first letter of $\overline{w}$ is the smallest μ such that $w_\mu \mu$ is a prefix of w.
- Erase the leftmost occurrence of μ in w and update all prefixes w_λ.

Example: $w = 341231$, with $12 = 21$, $14 = 41$, $23 = 32$ and $24 = 42$



$\overline{w}$: 2 3 1 4 3 1

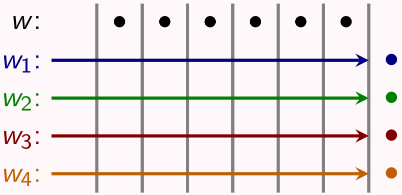
G – Swapping Places

Idea

Find the letters of $\overline{w}$ one by one, from left to right:

- For each letter λ, find the longest prefix w_λ of w that commutes with λ and does not contain λ.
- The first letter of $\overline{w}$ is the smallest μ such that $w_\mu\mu$ is a prefix of w.
- Erase the leftmost occurrence of μ in w and update all prefixes w_λ.

Example: $w = 341231$, with $12 = 21$, $14 = 41$, $23 = 32$ and $24 = 42$

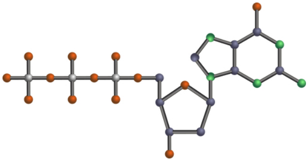


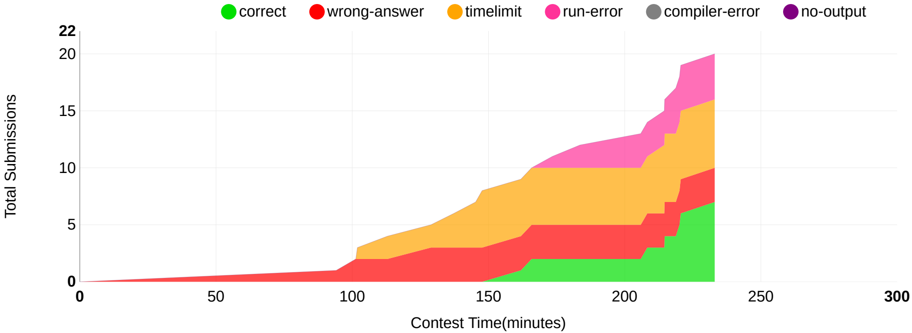
$\overline{w}$: 2 3 1 4 3 1

Time complexity: $\mathcal{O}(|A|\,|w|)$

D – Gnalcats

Solved by 8 teams before freeze.
First solved after 161 min by **mETH**.





D – Gnalcats

Problem

Stack language, inspired by Tezos' smart contract language Michelson.

Programs work on an infinite stack of values.

Values are either a pair of values or a non-pair.

Prove the equivalence of two programs on input stacks of non-pair values.

Instructions

COPY	Copy top value (`DUP`)
DROP	Drop top value (`DROP`)
SWAP	Swap top two values (`SWAP`)
PAIR	Construct pair from top two values (`PAIR`)
UNPAIR	Destruct top pair (`UNPAIR`), **FAIL** on non-pair values
LEFT	Replace top pair by its left component (`CAR`) $\equiv$ **USD**
RIGHT	Replace top pair by its right component (`CDR`) $\equiv$ **UD**

D – Gnalcats

Solution

Symbolic evaluation

- give a unique identifier to elements of the input stack (first $10^5 + 2$ elements are enough)
- evaluate both programs on this symbolic input stack (linear in program size)
- compare symbolic output stacks (linear in output stack overall sizes)

But...

Values can grow exponentially!

E.g. PAIR COPY PAIR COPY PAIR COPY ...

D – Gnalcats

But...

Values can grow exponentially! E.g. PAIR COPY PAIR COPY ...

Solution

Hash-consing

- give the same identifier to all pairs constructed from the same elements
- use a hash table $\langle \mathit{left\_id}, \mathit{right\_id} \rangle \rightarrow \mathit{pair\_id}$

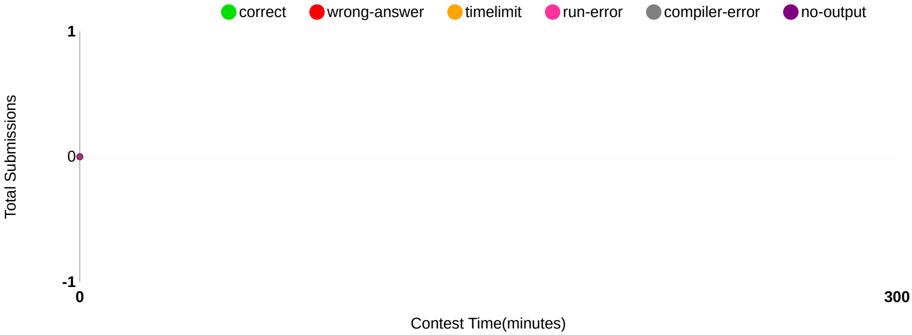
Complexity of comparison becomes linear in the size of stacks ($\leq 10^5$).

Even better (not necessary here)

- Represent stacks as pairs $\langle \mathit{top}, \mathit{rest} \rangle$
- Allows comparison in $\mathcal{O}(1)$

Or worse: use congruence-closure with a union-find

Not solved before freeze.



E – Pixels

Problem

You are given:

- a grid g of black/white pixels: all pixels are white at start;
- a family of controllers: pressing c switches the pixels in a set S_c;
- a target grid t.

Can you draw the grid t? If yes, by pressing which controllers?

Limits

- g and t can be long: $|g| = |t| = KL \leqslant 100\,000$;
- for each controller, $|S_c| \leqslant 5$;
- controllers are arranged along a grid: sets S_c are **very** regular.

We can work in time $\mathcal{O}(\min\{K, L\}KL) \leqslant \mathcal{O}((KL)^{3/2})$ but not $\Omega((KL)^2)$.

E – Pixels

Idea: Reduce the problem to solving a linear equation in $\mathbb{F}_2^{KL}$

- One pixel = one element of $\mathbb{F}_2$
- Grids v and t = vectors in $\mathbb{F}_2^{KL}$
- Family of sets S_c = sparse $(KL) \times (KL)$ matrix M
- Pressing a set C of controllers = Obtaining the vector $M \cdot C$

Solution

Use Gaussian elimination, starting from the controllers associated with top-left pixels, and find a C such that $M \cdot C = t$ (if any).

Time complexity: $\mathcal{O}(\min\{K, L\}KL)$