GCPC 2020
Presentation of solutions

Jury and Testers

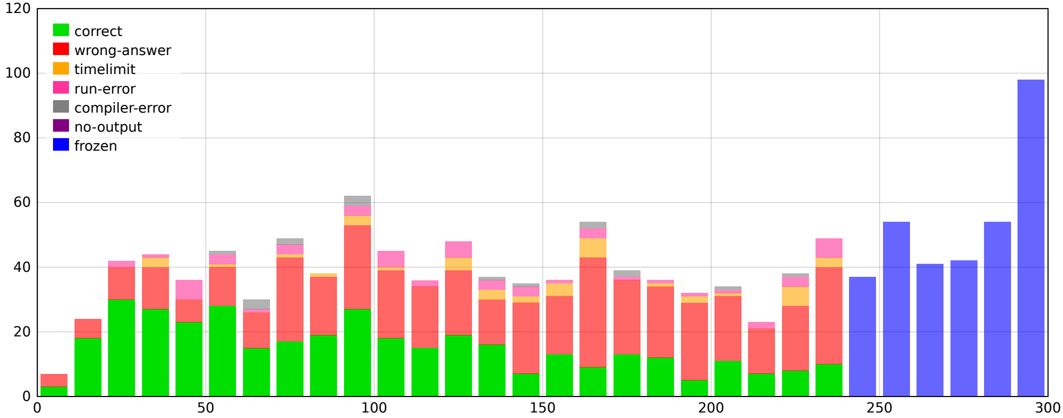
Thanks to the jury:

- Michael Baer (FAU)
- Julian Baldus (UdS)
- Gregor Behnke (Freiburg)
- Sandro Esquivel (CAU)
- Maximilian Fichtl (TUM)
- Nathan Maier (Ulm)
- Tobias Meggendorfer (TUM)
- Philipp Reger (FAU)
- Gregor Schwarz (TUM)
- Paul Wild (FAU)

Thanks to our test readers:

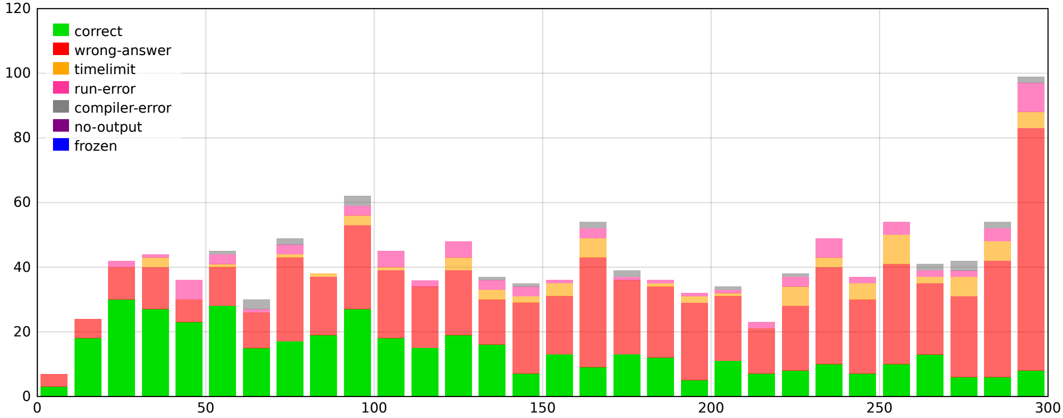
- Gregor Matl (TUM)
- Marcel Wienöbst (Lübeck)

Statistics



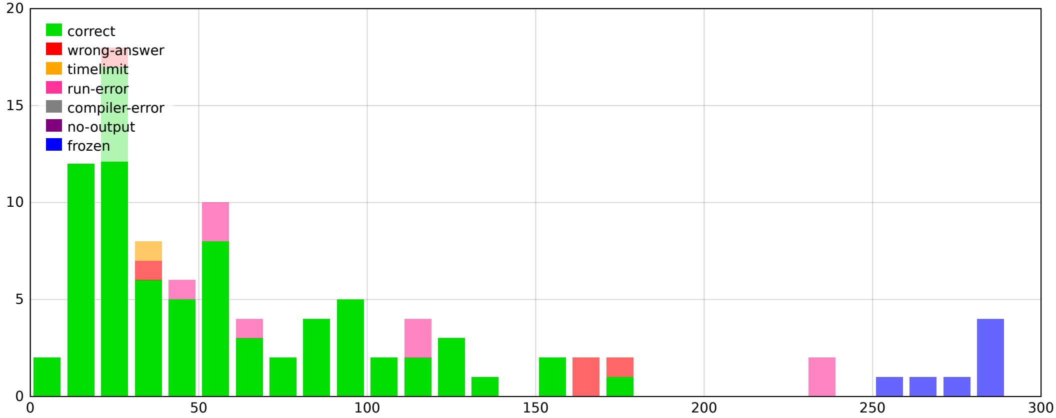
120
100
80
60
40
20
0
0
50
100
150
200
250
300
correct
wrong-answer
timelimit
run-error
compiler-error
no-output
frozen

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

F – Flip Flow



correct
wrong-answer
timelimit
run-error
compiler-error
no-output
frozen
0
5
10
15
20
0
50
100
150
200
250
300

F – Flip Flow

Problem

Given a list of times at which an hourglass is flipped over, how much sand remains on the upper half at the end of this process?

F – Flip Flow

Problem

Given a list of times at which an hourglass is flipped over, how much sand remains on the upper half at the end of this process?

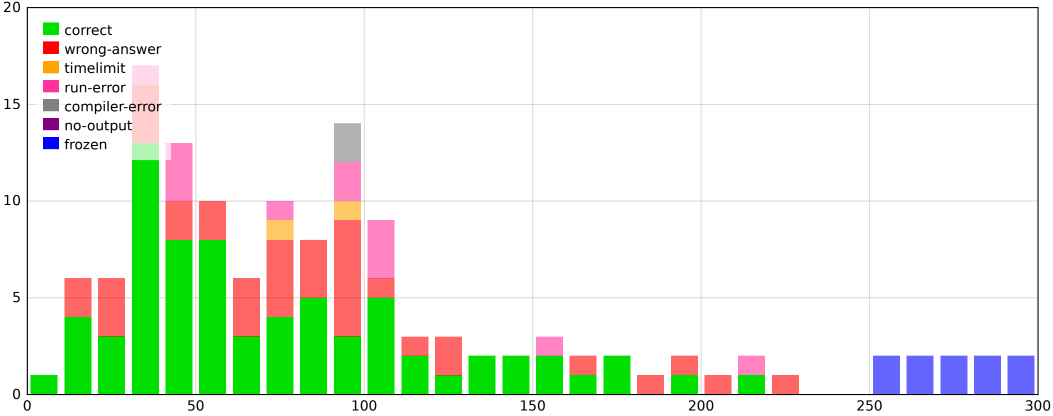
Solution

The process finishes after at most 10^6 seconds, so simulate step by step:

```
int upper = 0, lower = s;
for (int k = 0; k < t; k++) {
    if (flip[k]) swap(upper,lower);
    if (upper > 0) upper--, lower++;
}
```

Can also be solved in $\mathcal{O}(n)$, where n is the number of flips.

A – Adolescent Architecture



A – Adolescent Architecture

Problem

Given some cubical and cylindrical toy blocks, can they all be stacked into a single tower without there being any overhangs?

A – Adolescent Architecture

Problem

Given some cubical and cylindrical toy blocks, can they all be stacked into a single tower without there being any overhangs?

Solution

- If a solution exists, the footprint area of the blocks must always decrease towards the top of the tower.

A – Adolescent Architecture

Problem

Given some cubical and cylindrical toy blocks, can they all be stacked into a single tower without there being any overhangs?

Solution

- If a solution exists, the footprint area of the blocks must always decrease towards the top of the tower.
- Sort the blocks by area and check for each adjacent pair if one fits inside the other.

A – Adolescent Architecture

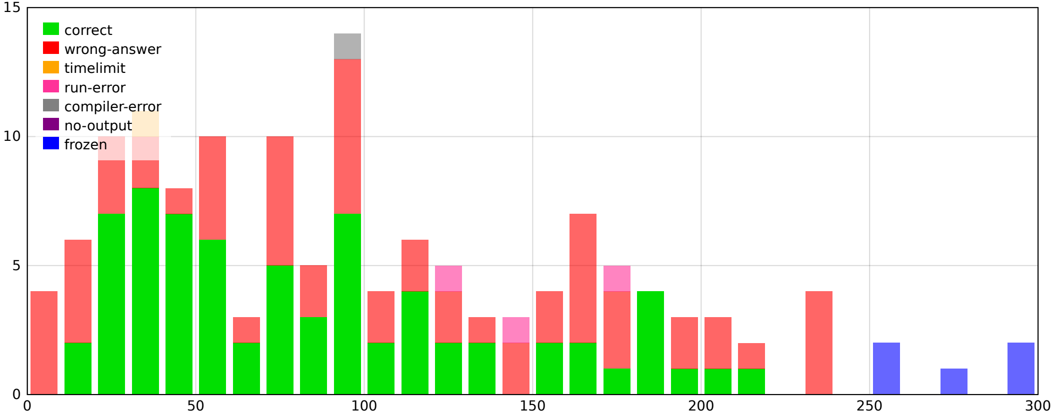
Problem

Given some cubical and cylindrical toy blocks, can they all be stacked into a single tower without there being any overhangs?

Solution

- If a solution exists, the footprint area of the blocks must always decrease towards the top of the tower.
- Sort the blocks by area and check for each adjacent pair if one fits inside the other.
- There is a simple formula for each of the four cases.

M – Mixtape Management



M – Mixtape Management

Problem

Given a permutation $p_1, \ldots, p_n$, find a sequence of positive integers $a_1, \ldots, a_n$ where $\text{str}(a_1) < \cdots < \text{str}(a_n)$ lexicographically and $a_{p_1} < \cdots < a_{p_n}$ by value.

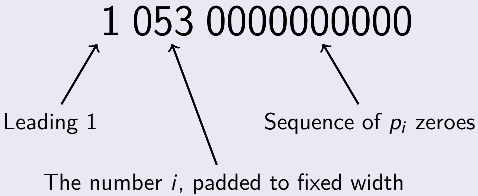
M – Mixtape Management

Problem

Given a permutation $p_1, \ldots, p_n$, find a sequence of positive integers $a_1, \ldots, a_n$ where $\text{str}(a_1) < \cdots < \text{str}(a_n)$ lexicographically and $a_{p_1} < \cdots < a_{p_n}$ by value.

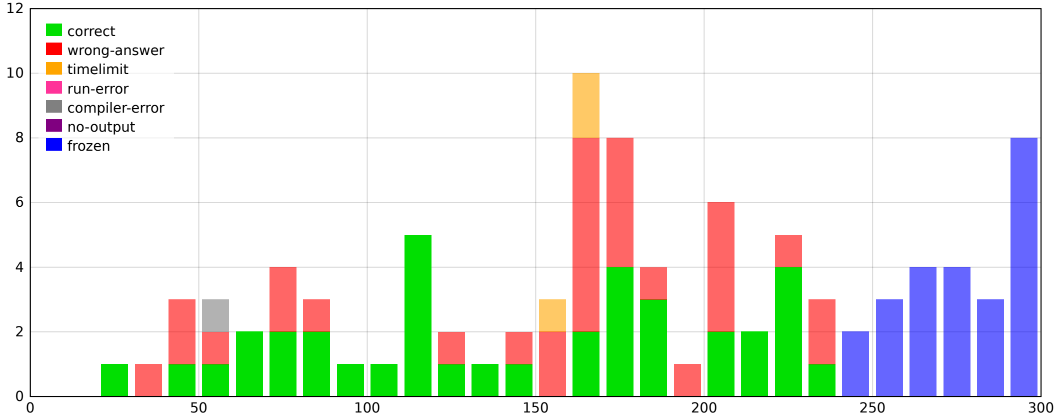
Solution

- Compose every number from three parts:



- This guarantees that all numbers are valid and the sorting is correct in both cases.

C – Confined Catching



C – Confined Catching

Problem

You are playing a board game on a square grid. You have two pieces while your AI opponent has one. Catch the opponent's piece within a limited number of turns.

C – Confined Catching

Problem

You are playing a board game on a square grid. You have two pieces while your AI opponent has one. Catch the opponent's piece within a limited number of turns.

Solution

Obviously, you have to move your pieces towards the AI's.

C – Confined Catching

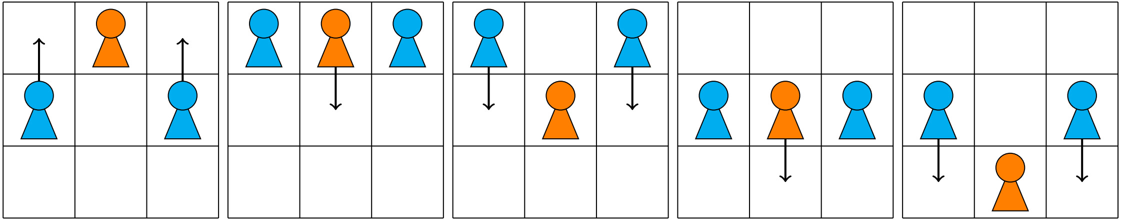
Solution cont.

However, just reducing the distance in all your turns may not be enough, as the AI may be able to keep fleeing forever.

C – Confined Catching

Solution cont.

However, just reducing the distance in all your turns may not be enough, as the AI may be able to keep fleeing forever.



C – Confined Catching

Solution cont.

The crucial strategy is to have your two pieces behave slightly differently:

- For your first piece, if it could move along either axis to get closer to the opponent, move along the y axis first.
- For your second piece, prioritize the x axis.

Eventually, the AI will be forced into a corner (or even lose before that), with nowhere left to run.

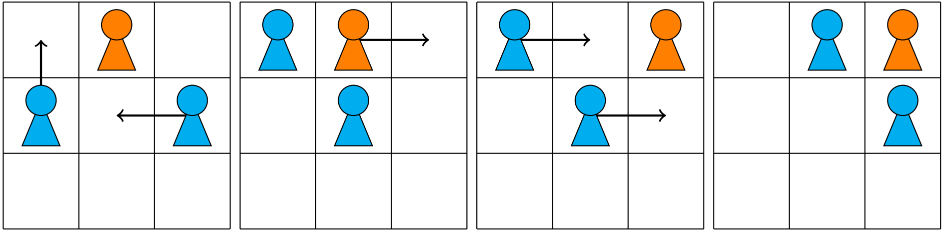
C – Confined Catching

Solution cont.

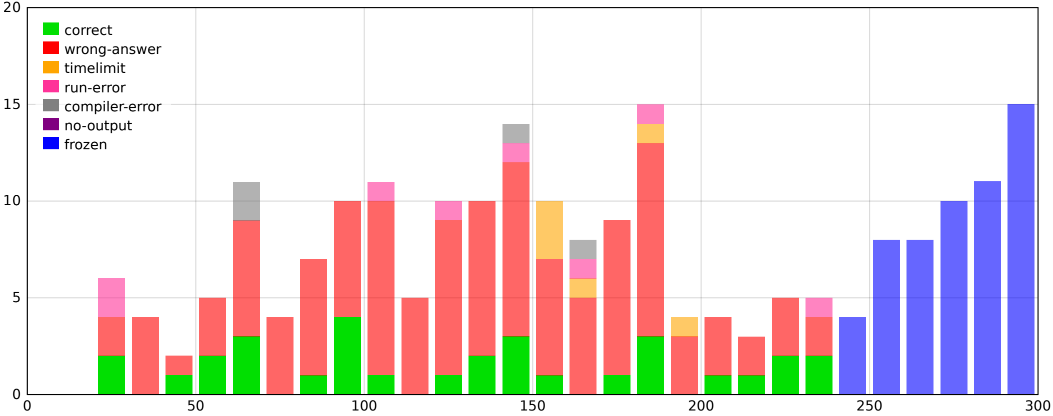
The crucial strategy is to have your two pieces behave slightly differently:

- For your first piece, if it could move along either axis to get closer to the opponent, move along the y axis first.
- For your second piece, prioritize the x axis.

Eventually, the AI will be forced into a corner (or even lose before that), with nowhere left to run.



B – Bookshelf Building



B – Bookshelf Building

Problem

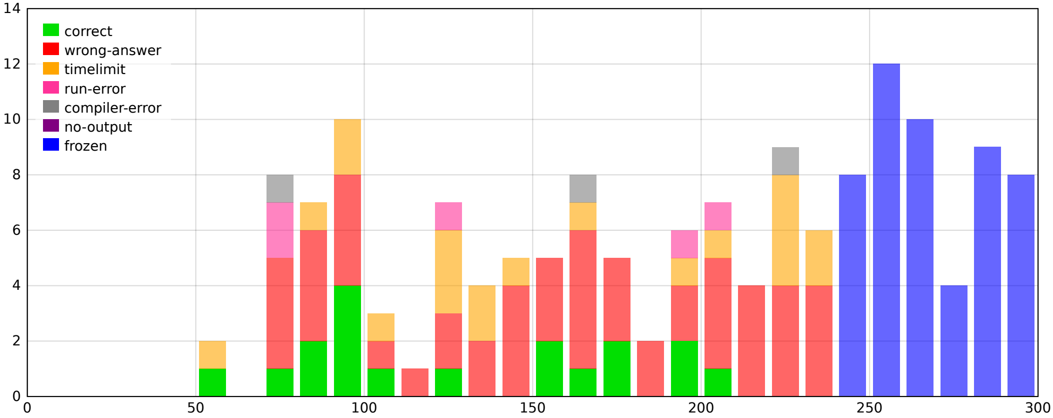
Given n books of different widths and heights, can you fit them into a rectangular bookshelf using at most one separating board?

B – Bookshelf Building

Solution

- Place the tallest book in the bottom left corner of the shelf.
- Install the board at the height of the tallest book.
- Greedily place all books in the lower section of the shelf that do not fit in the upper section.
- For the remaining capacity in the lower section, solve a knapsack problem – the more you fit into the lower section, the more space you have left in the upper section.
- Place all remaining books in the upper section.
- Special case: install no board if tallest book has height y
- Complexity: $\mathcal{O}(n\,x)$

G – Gravity Grid



G – Gravity Grid

Problem

Alice and Bob are playing a version of Connect Four on an $h \times w$ board where the goal is to complete a row of k tiles. Given a log of their moves, determine the winner.

G – Gravity Grid

Problem

Alice and Bob are playing a version of Connect Four on an $h \times w$ board where the goal is to complete a row of k tiles. Given a log of their moves, determine the winner.

Solution

- For each starting cell and each of the eight directions store the length of the longest run of equal tiles in that direction. Update these values as new tiles drop.

G – Gravity Grid

Problem

Alice and Bob are playing a version of Connect Four on an $h \times w$ board where the goal is to complete a row of k tiles. Given a log of their moves, determine the winner.

Solution

- For each starting cell and each of the eight directions store the length of the longest run of equal tiles in that direction. Update these values as new tiles drop.
- After each drop, it is enough to update the values in the current cell and the opposite cell of the run in each direction.

G – Gravity Grid

Problem

Alice and Bob are playing a version of Connect Four on an $h \times w$ board where the goal is to complete a row of k tiles. Given a log of their moves, determine the winner.

Solution

- For each starting cell and each of the eight directions store the length of the longest run of equal tiles in that direction. Update these values as new tiles drop.
- After each drop, it is enough to update the values in the current cell and the opposite cell of the run in each direction.
- Time and space complexity: $\mathcal{O}(h \cdot w)$.

G – Gravity Grid

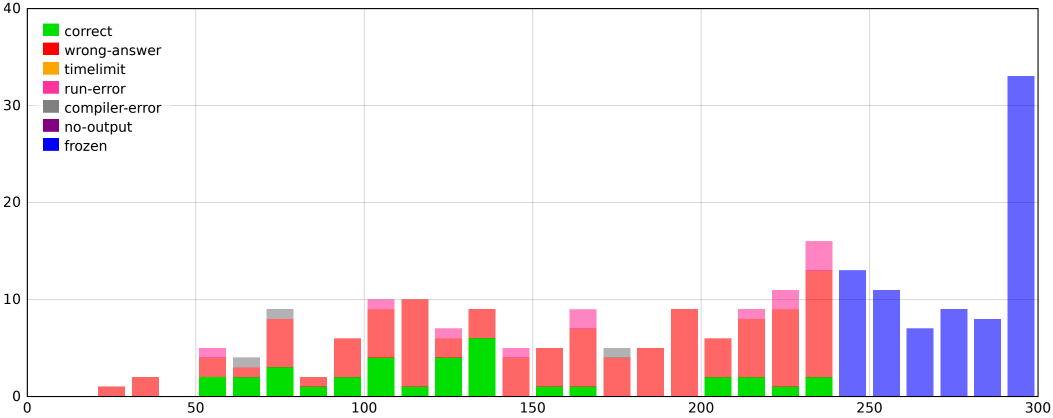
Problem

Alice and Bob are playing a version of Connect Four on an $h \times w$ board where the goal is to complete a row of k tiles. Given a log of their moves, determine the winner.

Solution

- For each starting cell and each of the eight directions store the length of the longest run of equal tiles in that direction. Update these values as new tiles drop.
- After each drop, it is enough to update the values in the current cell and the opposite cell of the run in each direction.
- Time and space complexity: $\mathcal{O}(h \cdot w)$.
- Several other solutions are possible, for instance using binary search and a two-pointer method or using a monotone queue.

K – Knightly Knowledge



K – Knightly Knowledge

Problem

Given are the coordinates of *monuments* and *churches*. Churches with ≥ 2 mon. in their row / col are *mighty*. Place one monument to maximize the number of churches that are turned mighty.

K – Knightly Knowledge

Problem

Given are the coordinates of *monuments* and *churches*. Churches with ≥ 2 mon. in their row / col are *mighty*. Place one monument to maximize the number of churches that are turned mighty.

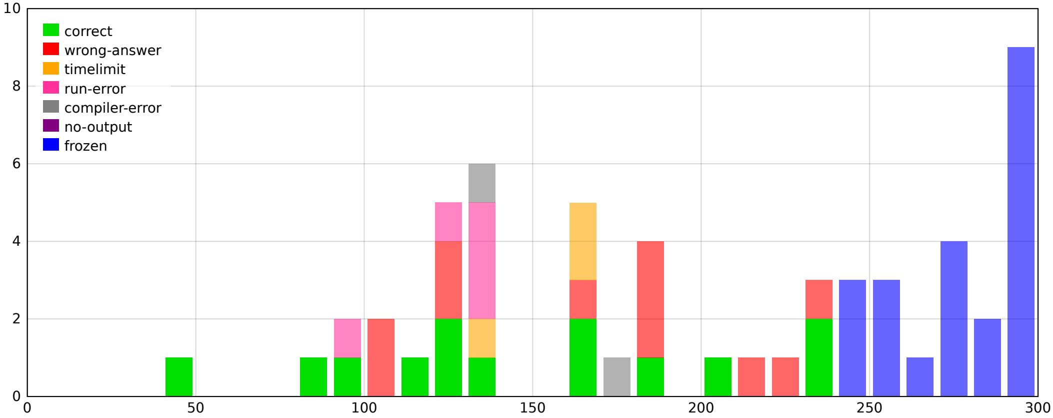
Solution $O((m+c)^2)$

- Count monuments and ordinary churches per row / column.
- Iterate over all reasonable locations to find the best spot.
- *Reasonable:* At least on church or monument in the same row and column.
- Do *not* count a church at the intersection twice.

Solution $O(m+c)$

- As above, but only check best row / col. They are either optimal or off by one.
- $\rightsquigarrow$ No other intersection can be better.
- Each non-optimal intersection has a church at the spot $\rightsquigarrow$ at most c.

D – Decorative Dominoes



D – Decorative Dominoes

Problem

Given an arrangement of dominoes without numbers, assign numbers to both halves of each domino such that

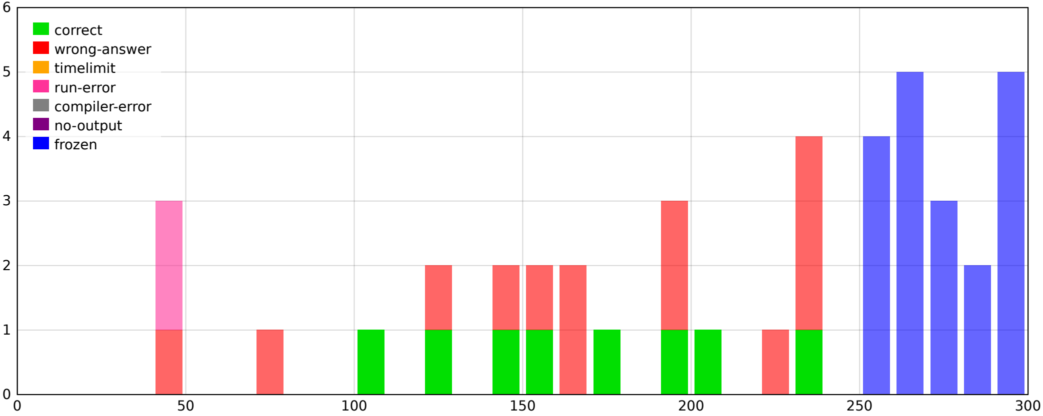
- each domino half is adjacent to a half of another domino with the same number on it;
- all numbers appear at most twice among all dominoes.

D – Decorative Dominoes

Solution

- Transform the problem into a graph where the nodes are the domino halves and the edges exist between adjacent halves belonging to different dominoes.
- The graph is bipartite: Imagine a large black and white checkered board over the coordinate grid. Connected nodes must have different colors.
- Find a perfect bipartite matching on this graph.
- Matched nodes are assigned the same number.
- Every valid numbering corresponds to a perfect matching.
  So whenever a solution exists, this algorithm will find one.
- Complexity: $\mathcal{O}(n^2)$

I – Impressive Integers



I – Impressive Integers

Problem

For a given integer n, determine if there exist integers a, b, and c such that an equilateral triangle with side length c can be tiled with exactly n triangles with side lengths a or b.

If possible, output a valid tiling.

I – Impressive Integers

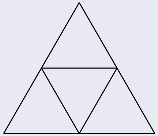
Solution

- By trying out small numbers one can find that it is impossible for $n = 2, 3, 5$.
- For all other $n > 0$ a valid tiling can be found as follows:

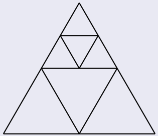
  $n = 1$ $a = b = c$

  $n > 2$ is even Use pattern 1a with $n - 1$ triangles in the bottom row.

  $n > 5$ is odd Use pattern 1b with $n - 4$ triangles in the bottom row.



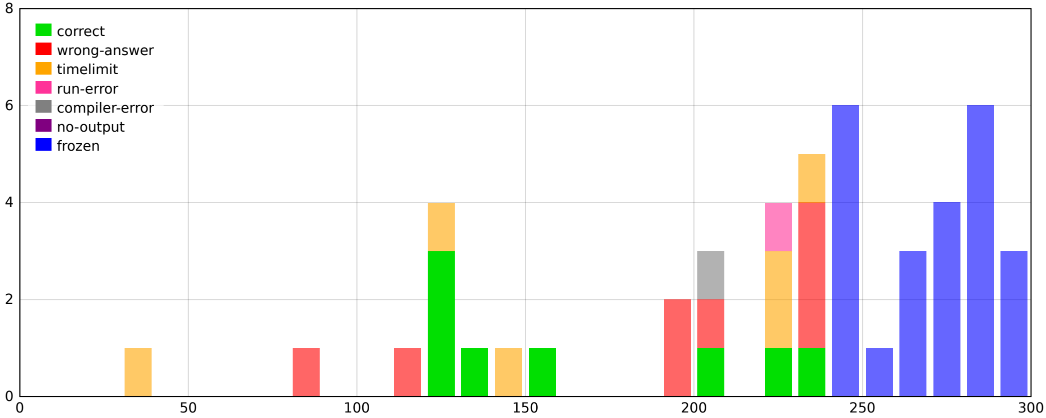
(a) Even pattern.



(b) Odd pattern.

- Complexity: $\mathcal{O}(n)$

L – Lexicographical Lecturing



L – Lexicographical Lecturing

Problem

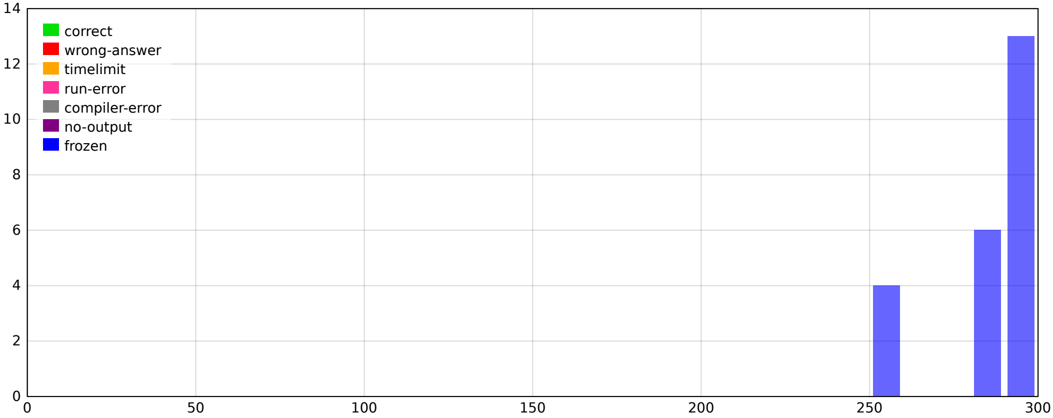
Find indices $i < j$ describing minimal-length substrings, such that the order with respect to the substring is equal the the original order.

L – Lexicographical Lecturing

Solution

- If any two subsequent strings s_k, s_{k+1} are ordered correctly w.r.t. interval (i, j), then all strings can be ordered correctly w.r.t. (i, j).
- Consider all subsequent strings s_k, s_{k+1} one after another.
- For each index i, let α_{ki} be the smallest index such that s_k and s_{k+1} are sorted correctly w.r.t. interval (i, α_{ki}).
  If no such index exists, set $\alpha_{ki} = \infty$.
- Determining all α_{ki} for two subsequent strings can be done in $\mathcal{O}(\ell)$ using dynamic programming.
- For each index i, determine the maximum α_{ki} over all k.
- Output the shortest interval among all $(i, \max_k\{\alpha_{ki}\})$.
- Complexity: $\mathcal{O}(n\,\ell)$

J – Jeopardised Journey



J – Jeopardised Journey

Problem

Given points (glades) and circles (hills). You can go from point A to B iff the direct line between them does not intersect a circle or any other point.
Opponent selects (unknown) one point to block. Determine for a starting point, which other points can be reached no matter which point is blocked by the opponent.

J – Jeopardised Journey

Problem

Given points (glades) and circles (hills). You can go from point A to B iff the direct line between them does not intersect a circle or any other point.
Opponent selects (unknown) one point to block. Determine for a starting point, which other points can be reached no matter which point is blocked by the opponent.

Solution

Two part problem:

- Geometry: Determine for which points A and B there is no circle/other point between them and build a graph.
- Graph: Find all nodes of the graph with two fully disjunct paths to node 0.

J – Jeopardised Journey

Solution: Geometry

1000 points and 1000 circles $\Rightarrow$ We can't test all pairs of points (would be $\mathcal{O}(n^3)$).

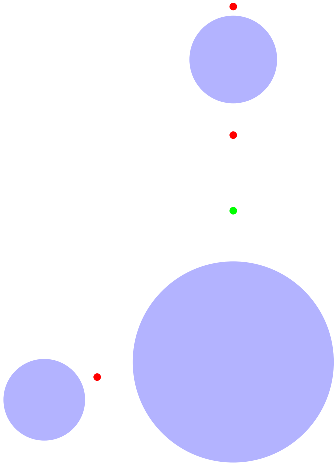
J – Jeopardised Journey

Solution: Geometry

1000 points and 1000 circles $\Rightarrow$ We can't test all pairs of points (would be $\mathcal{O}(n^3)$).

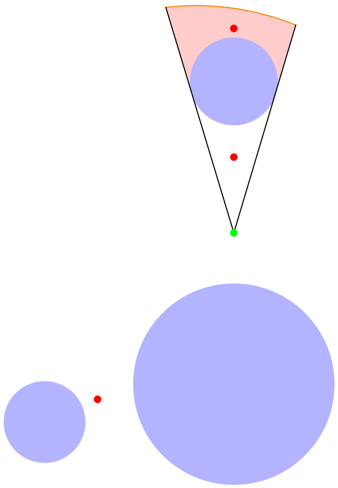
Angular Sorting per point.

J – Jeopardised Journey

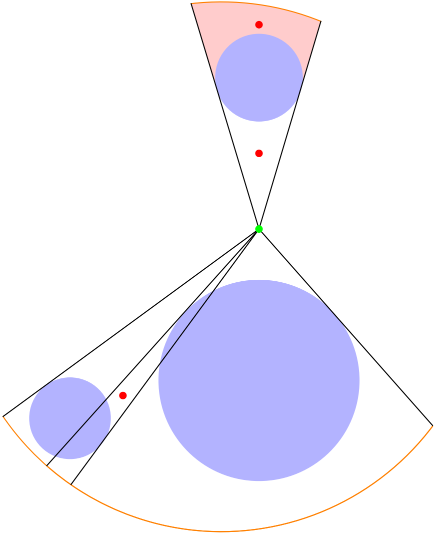


- Consider one point (green).

J – Jeopardised Journey

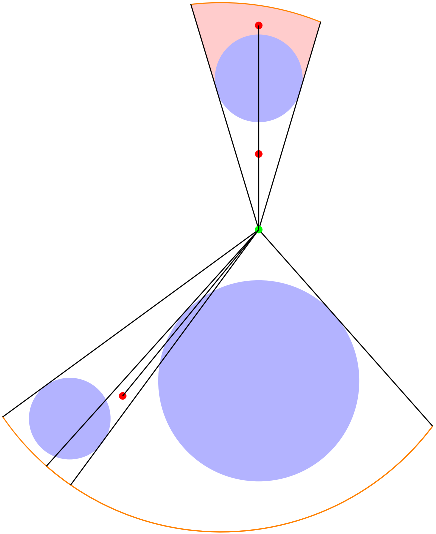


- Consider one point (green).
- A Circle "excludes" all points behind it.



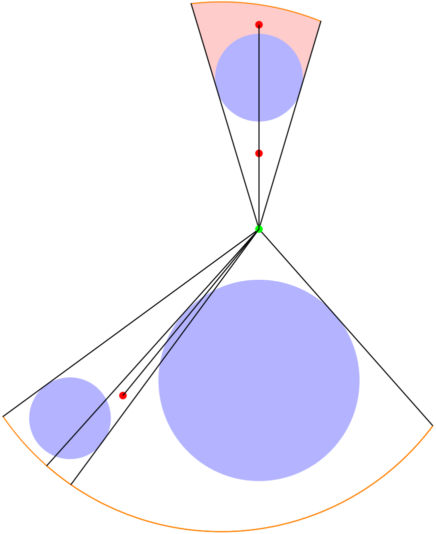
- Consider one point (green).
- A Circle "excludes" all points behind it.
- Compute angles of tangents of circles and the current point.

J – Jeopardised Journey



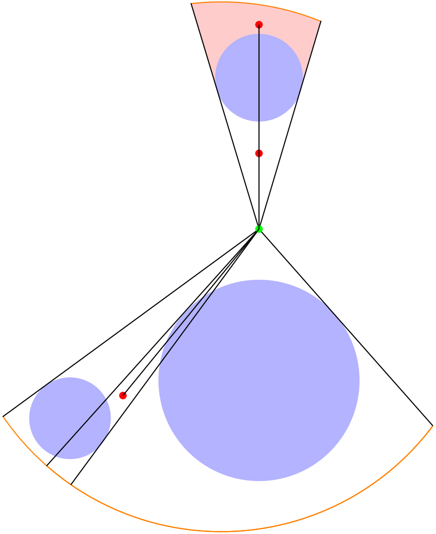
- Consider one point (green).
- A Circle "excludes" all points behind it.
- Compute angles of tangents of circles and the current point.
- Compute angles between the current point and all other points.

J – Jeopardised Journey



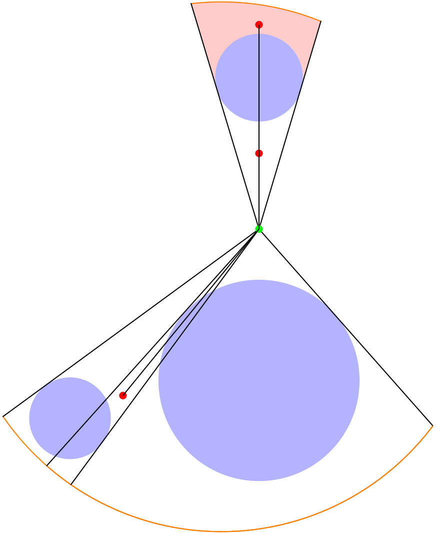
- Consider one point (green).
- A Circle "excludes" all points behind it.
- Compute angles of tangents of circles and the current point.
- Compute angles between the current point and all other points.
- Sort these ≤ 2999 events (start and end per circle and points)

J – Jeopardised Journey



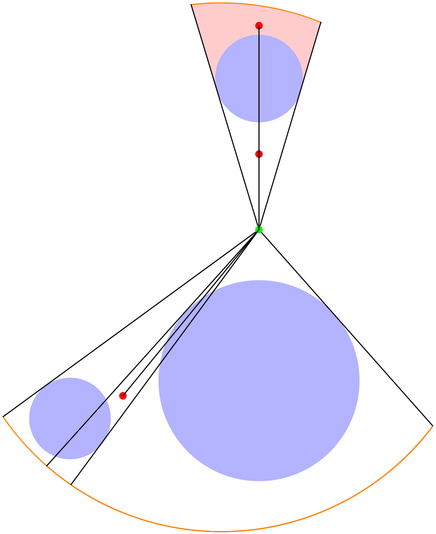
- Process events in angular order
  - Maintain set S of currently started, but not finished circles (with the distance of the tangent points)
  - If next event is

J – Jeopardised Journey



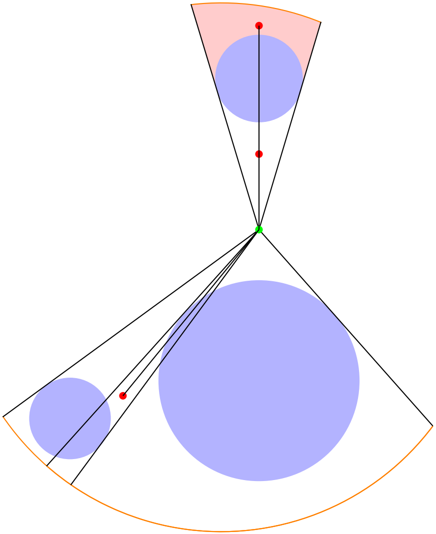
- Process events in angular order
  - Maintain set S of currently started, but not finished circles (with the distance of the tangent points)
  - If next event is
    - Start circle: add to S

J – Jeopardised Journey



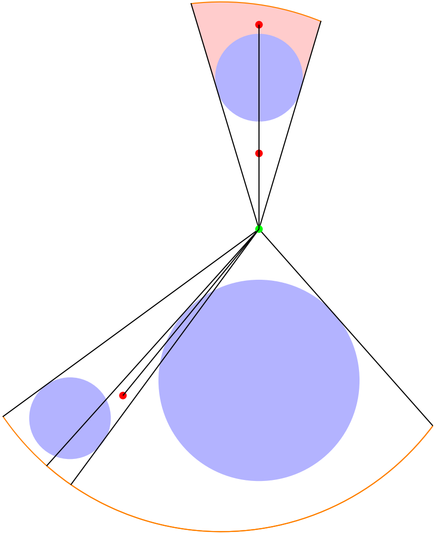
- Process events in angular order
  - Maintain set S of currently started, but not finished circles (with the distance of the tangent points)
  - If next event is
    - Start circle: add to S
    - End circle: remove from S

J – Jeopardised Journey



- Process events in angular order
  - Maintain set S of currently started, but not finished circles (with the distance of the tangent points)
  - If next event is
    - Start circle: add to S
    - End circle: remove from S
    - Point: is safe if all $s \in S$ have a higher distance than all tanget points (use a C++ multiset).

J – Jeopardised Journey



- Process events in angular order
  - Maintain set S of currently started, but not finished circles (with the distance of the tangent points)
  - If next event is
    - Start circle: add to S
    - End circle: remove from S
    - Point: is safe if all $s \in S$ have a higher distance than all tanget points (use a C++ multiset).
- Runtime: $\mathcal{O}(n \log n)$

J – Jeopardised Journey

Solution: Geometry Pitfalls

Implementation has a lot of pitfalls

- Multiple events with same angle: end circle – points in increasing distance – begin circle
- For multiple points with same angle: add edge only to the first one.
- Circles that intersect with the $(+, 0)$ axis have $\alpha_{begin} > \alpha_{end}$.
  $\Rightarrow$ split into two along angle 0.

J – Jeopardised Journey

Solution: Graph

Find all nodes of a graph $G = (V, E)$ for which two fully disjunct paths to node 0 exist.

Solution: Graph

Find all nodes of a graph $G = (V, E)$ for which two fully disjunct paths to node 0 exist.

$n \leq 2000$ so $\mathcal{O}(n^2)$ would be ok.

J – Jeopardised Journey

Solution: Graph

Find all nodes of a graph $G = (V, E)$ for which two fully disjunct paths to node 0 exist.

$n \leq 2000$ so $\mathcal{O}(n^2)$ would be ok.

Try every blocked vertex. Do DFS from node 0.
Count how often each vertex is reached. If it is reached in $|V| - 2$ DFSs (it is blocked once) then it is safe.
Graph contains n^2 edges, so this would have runtime n^3 (too slow).

J – Jeopardised Journey

Solution: Graph

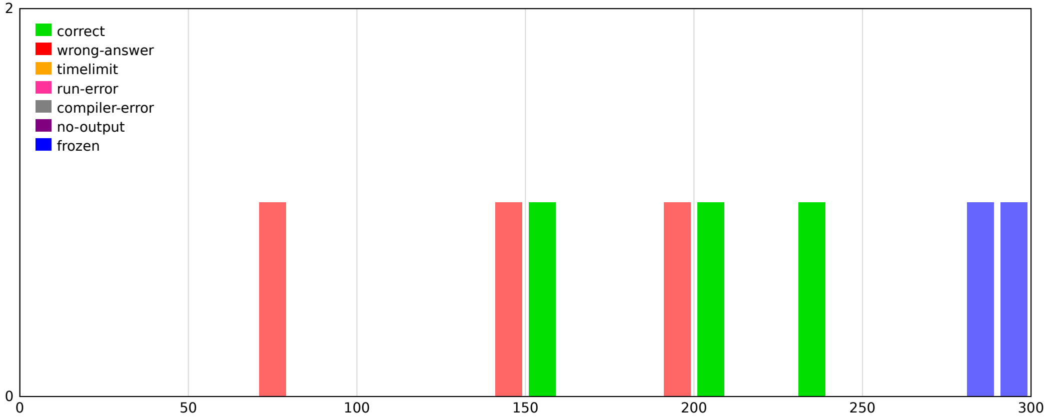
Find all nodes of a graph $G = (V, E)$ for which two fully disjunct paths to node 0 exist.

$n \leq 2000$ so $\mathcal{O}(n^2)$ would be ok.

Try every blocked vertex. Do DFS from node 0.
Count how often each vertex is reached. If it is reached in $|V| - 2$ DFSs (it is blocked once) then it is safe.
Graph contains n^2 edges, so this would have runtime n^3 (too slow).

Better solution: Articulation nodes (can be determined in linear time using DFS).
A node is safe if it is reachable from 0 without traversing an articulation node.
$2\times$ DFS $\Rightarrow O(n^2)$

E – Exhausting Errands



E – Exhausting Errands

Problem

Given n pairs of integers $(a_1, b_1), \ldots, (a_n, b_n)$, find the length of the shortest 1D route visiting all positions a_i, b_i subject to the constraint that a_i is visited before b_i. The route can start at any a_i and finish at any b_i.

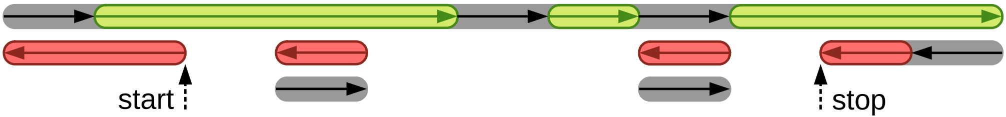
E – Exhausting Errands

Problem

Given n pairs of integers $(a_1, b_1), \ldots, (a_n, b_n)$, find the length of the shortest 1D route visiting all positions a_i, b_i subject to the constraint that a_i is visited before b_i. The route can start at any a_i and finish at any b_i.

Solution

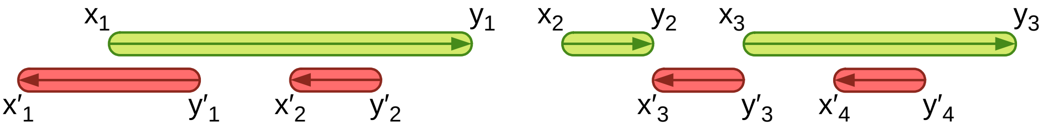
Idea: Complete all “forward“ errands during one left-to-right run, complete “backward” errands either before, after or during this run.



E – Exhausting Errands

Solution: Preparation

- Partition pairs into "forward" pairs ($a_i \leq b_i$) and "backward" pairs ($a_i > b_i$).
- Sort pairs within each partition ascending by start position.
- Merge all overlapping or adjoint pairs within each partition.
- Denote resulting m "forward" pairs as $(x_1, y_1), \ldots, (x_m, y_m)$ and m' "backward" pairs as $(x'_1, y'_1), \ldots, (x'_{m'}, y'_{m'})$.
- Note that $x_1 \leq y_1 < \ldots < x_m \leq y_m$ and $y'_1 < x'_1 < \ldots < y'_{m'} < x'_{m'}$.
- Complexity: $\mathcal{O}(n \log(n))$ for sorting, $\mathcal{O}(n)$ for merging



E – Exhausting Errands

Solution: Special Case 1

- Assume that $m' = 0$. Then the trivial solution is to start at x_1 and finish at y_m (single "forward pass"). The resulting distance is $y_m - x_1$.

E – Exhausting Errands

Solution: Special Case 1

- Assume that $m' = 0$. Then the trivial solution is to start at x_1 and finish at y_m (single "forward pass"). The resulting distance is $y_m - x_1$.

Solution: Special Case 2

- Assume that $m' = 1$. Then we have three options:
  - Go from x'_1 to y'_1 before the forward pass and proceed to x_1 afterwards. The extra distance is $x'_1 - y'_1 + |x_1 - y'_1|$.
  - Stop at x'_1 during the forward pass, go to y'_i and return to x'_i (only applicable if $x_1 < x'_1 < y_m$). The extra distance is $2(x'_1 - y'_1)$.
  - Process to x'_1 and y'_1 after the forward pass. The extra distance is $x'_1 - y'_1 + |x'_1 - y_m|$.

E – Exhausting Errands

Solution: General Case

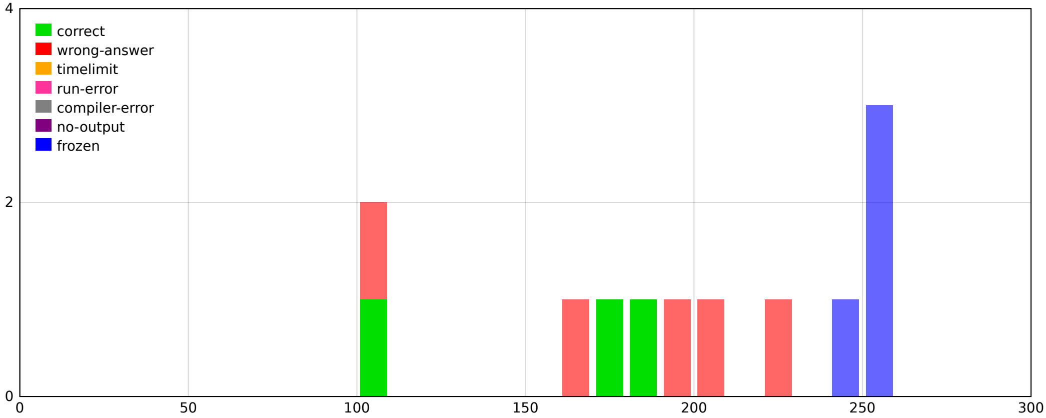
- Assume that $m' > 1$. Now we can complete the first s backward errands before the forward pass, the last t after and the remaining $m' - s - t$ backward errands during the forward pass $(0 \leq s \leq m', 0 \leq t \leq m' - s)$:
  - The extra distance for the first part is $x'_s - y'_1 + |x_1 - y'_1|$.
  - The extra distance for the inbetween part is $\sum_{i=s+1}^{m'-t} 2(x'_i - y'_i)$.
  - The extra distance for the last part is $x'_{m'} - y'_{m'-t+1} + |x'_{m'} - y_m|$.
- Check the total distance for all feasible s, t and pick the minimal solution.
- Complexity: $\mathcal{O}(n^2)$

E – Exhausting Errands

Solution: Caveats

- Solve also the mirrored problem, *i. e.*, for errands $(-a_1, -b_1), \ldots, (-a_n, -b_n)$, and pick its solution if better.
- Do not evaluate special cases $s = 0, t < m'$ when $x'_1 \leq x_1$ resp. $t = 0, s < m'$ when $x'_{m'} \geq y_m$ (*i. e.*, start points of some "inbetween" errands are outside of the forward pass).
- Use sum arrays for x'_i and y'_i to compute the extra distance for the inbetween part in $\mathcal{O}(1)$.

H – Hectic Harbour



H – Hectic Harbour

Problem

Schedule two gantry cranes such that they finish their assigned tasks as fast as possible.

H – Hectic Harbour

Solution

- Define two DP arrays for cranes A and B:

  dpA[i][j][p]: A finished task i, B finished task j.
  A is at position of task i, B is at position p.

  dpB[i][j][p]: A finished task i, B finished task j.
  A is at position p, B is at position of task j.

- Distinguish three cases when updating DP arrays:
  1. A and B both perform their next task if they need exactly the same number of steps. If not, consider case (2) or (3).
  2. A performs next task while B moves as close as possible towards its next task.
  3. B performs next task while A moves as close as possible towards its next task.
- Always ensure that cranes do not crash.
- Complexity: $\mathcal{O}(a\,b\,n)$
- Sweep line solutions in $\mathcal{O}(a\,b\,n\log(n))$ are also accepted.

Weiteres Programm

- Jetzt: Auflösung des Scoreboards und Siegerehrung
- Anschließend Voice-Chat auf dem Discord-Server
- Extended Contest mit den GCPC-Aufgaben (bald) unter

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

Danke für die Teilnahme!