NCPC 2023
Presentation of solutions

2023-10-07

Problems prepared by

- Arnar Bjarni Arnarson (Reykjavik University)
- Andreas Björklund (IT University of Copenhagen)
- Atli Fannar Franklín (University of Iceland)
- Nils Gustafsson (KTH Royal Institute of Technology)
- Oskar Haarklou Veileborg (Aarhus University)
- Asger Hautop Drewsen (Aarhus University)

Test solver

- Björn Martinsson

C — Converting Romans

Problem

Convert Roman numerals to regular Arabic numbers.

Solution

Statistics at 4-hour mark: 473 submissions, 154 accepted, first after HH:MM

C — Converting Romans

Problem

Convert Roman numerals to regular Arabic numbers.

Solution

1. Using a map/dictonary to map Roman digits to numbers is convenient.

Statistics at 4-hour mark: 473 submissions, 154 accepted, first after HH:MM

C — Converting Romans

Problem

Convert Roman numerals to regular Arabic numbers.

Solution

1. Using a map/dictonary to map Roman digits to numbers is convenient.
2. For each digit in a number, check if it is smaller than anything to the right. In that case, subtract it.

Statistics at 4-hour mark: 473 submissions, 154 accepted, first after HH:MM

C — Converting Romans

Problem

Convert Roman numerals to regular Arabic numbers.

Solution

1. Using a map/dictonary to map Roman digits to numbers is convenient.
2. For each digit in a number, check if it is smaller than anything to the right. In that case, subtract it.
3. Make sure this runs faster than $\mathcal{O}(L^2)$!

Statistics at 4-hour mark: 473 submissions, 154 accepted, first after HH:MM

C — Converting Romans

Problem

Convert Roman numerals to regular Arabic numbers.

Solution

1. Using a map/dictonary to map Roman digits to numbers is convenient.
2. For each digit in a number, check if it is smaller than anything to the right. In that case, subtract it.
3. Make sure this runs faster than $\mathcal{O}(L^2)$!
4. This can be done by looping from the end of the string, and keeping track of the largest digit seen so far.

Statistics at 4-hour mark: 473 submissions, 154 accepted, first after HH:MM

H — Heroes of Velmar

Problem

Given the description of a card game and a final state of the game, implement the rules for resolving which player won the match.

Solution

1. Process each player and location pair individually to apply abilities and add all the power levels of the cards together.
    1. If the card is Seraphina and k is the number of friendly cards at the location, add $k - 1$ power to the location.
    2. If the card is Thunderheart and the location has 4 friendly cards, add 6 power to the location.
    3. If the card is Zenith, and the location is the center location, add 5 power to the location.
2. Count the number of locations where each player wins.
3. If one of the players wins more locations than the other, that player wins.

D — Die Hard

Problem

Given the description of three dice, calculate whether any of them will roll higher than both of the remaining two with probability $\geq \frac{1}{2}$.

Solution

D — Die Hard

Problem

Given the description of three dice, calculate whether any of them will roll higher than both of the remaining two with probability $\geq \frac{1}{2}$.

Solution

1. There are N^2 possible outcomes when rolling two dice with N sides.

Statistics at 4-hour mark: 143 submissions, 285 accepted, first after HH:MM

D — Die Hard

Problem

Given the description of three dice, calculate whether any of them will roll higher than both of the remaining two with probability $\geq \frac{1}{2}$.

Solution

1. There are N^2 possible outcomes when rolling two dice with N sides.
2. $N = 6$, so just try all combinations of rolls.

Statistics at 4-hour mark: 143 submissions, 285 accepted, first after HH:MM

K — Karl Coder[1]

Problem

Karl has a string of size $2 \leq N \leq 10^{18}$. It is contained in a buffer of size $2N$ where the first N bytes are non-zero and the last N bytes are zero. You need to find N by reading from the buffer, without reading at an index which is out of bounds.
You can only use $2\lceil \log_2 N \rceil$ reads.

Solution

[1] https://www.youtube.com/watch?v=gWexE-YVn0s

K — Karl Coder[1]

Problem

Karl has a string of size $2 \leq N \leq 10^{18}$. It is contained in a buffer of size $2N$ where the first N bytes are non-zero and the last N bytes are zero. You need to find N by reading from the buffer, without reading at an index which is out of bounds.
You can only use $2\lceil \log_2 N \rceil$ reads.

Solution

1. Linear search uses too many queries.

[1] https://www.youtube.com/watch?v=gWexE-YVn0s

K — Karl Coder[1]

Problem

Karl has a string of size $2 \leq N \leq 10^{18}$. It is contained in a buffer of size $2N$ where the first N bytes are non-zero and the last N bytes are zero. You need to find N by reading from the buffer, without reading at an index which is out of bounds.
You can only use $2\lceil \log_2 N \rceil$ reads.

Solution

1. Linear search uses too many queries.
2. First idea: Use binary search instead of linear search to reduce number of queries.

[1] https://www.youtube.com/watch?v=gWexE-YVn0s

K — Karl Coder[1]

Problem

Karl has a string of size $2 \leq N \leq 10^{18}$. It is contained in a buffer of size $2N$ where the first N bytes are non-zero and the last N bytes are zero. You need to find N by reading from the buffer, without reading at an index which is out of bounds.
You can only use $2\lceil \log_2 N \rceil$ reads.

Solution

1. Linear search uses too many queries.
2. First idea: Use binary search instead of linear search to reduce number of queries.
    1. Problem 1: Uses too many queries if N is small.

[1] https://www.youtube.com/watch?v=gWexE-YVn0s

K — Karl Coder[1]

Problem

Karl has a string of size $2 \leq N \leq 10^{18}$. It is contained in a buffer of size $2N$ where the first N bytes are non-zero and the last N bytes are zero. You need to find N by reading from the buffer, without reading at an index which is out of bounds.
You can only use $2\lceil \log_2 N \rceil$ reads.

Solution

1. Linear search uses too many queries.
2. First idea: Use binary search instead of linear search to reduce number of queries.
    1. Problem 1: Uses too many queries if N is small.
    2. Problem 2: Will read out of bounds if N is small.

[1] https://www.youtube.com/watch?v=gWexE-YVn0s

K — Karl Coder

Solution

1. First idea: Use binary search instead of linear search to reduce number of queries.

K — Karl Coder

Solution

1. First idea: Use binary search instead of linear search to reduce number of queries.
2. Second idea: First do exponential search (with base 2) to "safely" find an index i, where $N \leq i < 2N$. Then do binary search to find N in the range $\left[\frac{i+1}{2}, i\right]$.

K — Karl Coder

Solution

1. First idea: Use binary search instead of linear search to reduce number of queries.
2. Second idea: First do exponential search (with base 2) to “safely” find an index i, where $N \leq i < 2N$. Then do binary search to find N in the range $\left[\frac{i+1}{2}, i\right]$.
3. This can be done with $2\lceil\log_2 N\rceil$ reads in total.

K — Karl Coder

Solution

1. First idea: Use binary search instead of linear search to reduce number of queries.
2. Second idea: First do exponential search (with base 2) to "safely" find an index i, where $N \leq i < 2N$. Then do binary search to find N in the range $\left[\frac{i+1}{2}, i\right]$.
3. This can be done with $2\lceil \log_2 N \rceil$ reads in total.
4. You need to be careful to avoid off-by-one errors and to not use too many queries when N is small.

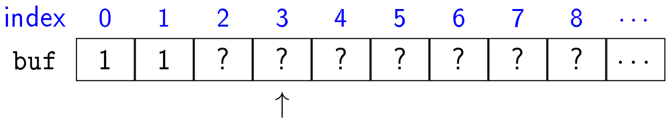
K — Karl Coder

$$2 \leq N \leq 10^{18}$$

index	0	1	2	3	4	5	6	7	8	...
buf	1	1	?	?	?	?	?	?	?	...

K — Karl Coder

$$2 \le N \le 10^{18}$$



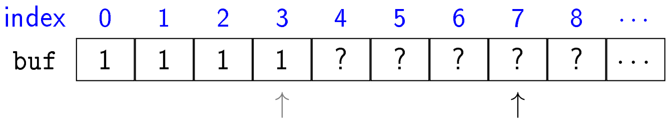
K — Karl Coder

$$4 \leq N \leq 10^{18}$$

index	0	1	2	3	4	5	6	7	8	$\cdots$
`buf`	1	1	1	1	?	?	?	?	?	$\cdots$
				$\uparrow$						

K — Karl Coder

$$4 \leq N \leq 10^{18}$$



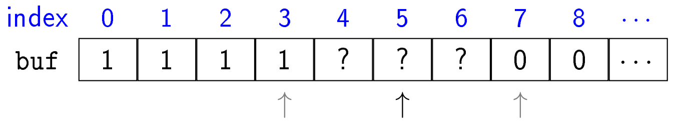
K — Karl Coder

$4 \leq N \leq 7$

index	0	1	2	3	4	5	6	7	8	$\cdots$
buf	1	1	1	1	?	?	?	0	0	$\cdots$
				$\uparrow$				$\uparrow$		

K — Karl Coder

$$4 \leq N \leq 7$$



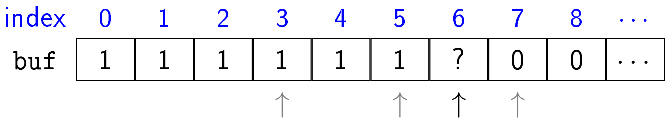
K — Karl Coder

$$6 \leq N \leq 7$$

index	0	1	2	3	4	5	6	7	8	$\cdots$
buf	1	1	1	1	1	1	?	0	0	$\cdots$
				↑		↑		↑		

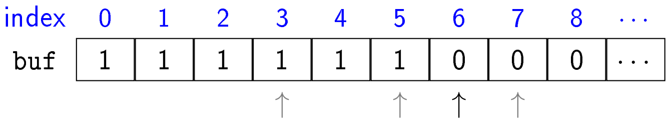
K — Karl Coder

$$6 \leq N \leq 7$$



K — Karl Coder

$N = 6$



K — Karl Coder

Solution

1. First idea: Use binary search instead of linear search to reduce number of queries.
2. Second idea: First do exponential search (with base 2) to “safely” find an index i, where $N \leq i < 2N$. Then do binary search to find N in the range $\left[\frac{i+1}{2}, i\right]$.
3. This can be done with $2\lceil \log_2 N \rceil$ reads in total.
4. You need to be careful to avoid off-by-one errors and to not use too many queries when N is small.

Statistics at 4-hour mark: 393 submissions, 115 accepted, first after HH:MM

J — Jamboree

Problem

Partition N items among M scouts so that no one has more than two items and the maximum total size is minimised.

Solution

Statistics at 4-hour mark: 442 submissions, 163 accepted, first after HH:MM

J — Jamboree

Problem

Partition N items among M scouts so that no one has more than two items and the maximum total size is minimised.

Solution

1. Sort the N items after decreasing size.

Statistics at 4-hour mark: 442 submissions, 163 accepted, first after HH:MM

J — Jamboree

Problem

Partition N items among M scouts so that no one has more than two items and the maximum total size is minimised.

Solution

1. Sort the N items after decreasing size.
2. If $N \leq M$, the answer is the size of largest item.

Statistics at 4-hour mark: 442 submissions, 163 accepted, first after HH:MM

J — Jamboree

Problem

Partition N items among M scouts so that no one has more than two items and the maximum total size is minimised.

Solution

1. Sort the N items after decreasing size.
2. If $N \leq M$, the answer is the size of largest item.
3. If $N > M$, pair up the $M+1$:th largest item with the M:th largest, the $M+2$:th largest item with the $M-1$:th largest, and so on until we have M groups of pairs and single elements. Report the maximum size.

Statistics at 4-hour mark: 442 submissions, 163 accepted, first after HH:MM

J — Jamboree

Problem

Partition N items among M scouts so that no one has more than two items and the maximum total size is minimised.

Solution

1. Sort the N items after decreasing size.
2. If $N \leq M$, the answer is the size of largest item.
3. If $N > M$, pair up the $M+1$:th largest item with the M:th largest, the $M+2$:th largest item with the $M-1$:th largest, and so on until we have M groups of pairs and single elements. Report the maximum size.
4. This greedy strategy works since for any four sizes $a \leq b \leq c \leq d$, the best way to divide them in two pairs minimising the maximum is $a+d$ and $b+c$.

Statistics at 4-hour mark: 442 submissions, 163 accepted, first after HH:MM

A — Aperiodic Appointments

Problem

A string is generated by repeatedly appending a 1 if the last characters contain a pattern that repeats K times, and a 0 otherwise. Count the number of ones.

A — Aperiodic Appointments

Problem

A string is generated by repeatedly appending a 1 if the last characters contain a pattern that repeats K times, and a 0 otherwise. Count the number of ones.

Solution

1. A good idea is to write out the string for $K = 3$ or 4 on paper, and look for patterns.

A — Aperiodic Appointments

```
000100010001100010001000110001000100011111111111
```

A — Aperiodic Appointments

$$\underline{\overline{0001}\,\overline{0001}\,\overline{0001}1}\;\underline{0001000100011}\;\underline{0001000100011}11111111$$

A — Aperiodic Appointments

Solution

1. Guess: the string is generated by first letting $s = 0$, and then applying $s = s * k + 1$ k times, and then append infinitely many ones.

Statistics at 4-hour mark: 379 submissions, 68 accepted, first after HH:MM

A — Aperiodic Appointments

Solution

1. Guess: the string is generated by first letting $s = 0$, and then applying $s = s * k + 1$ k times, and then append infinitely many ones.
2. There are now some recursive ways of counting the number of ones in $\mathcal{O}(\log(N))$.

Statistics at 4-hour mark: 379 submissions, 68 accepted, first after HH:MM

F — Factor-Full Tree

Problem

Given a rooted tree, label the vertices with positive integers so that the divisor graph equals the rooted tree.

Solution

F — Factor-Full Tree

Problem

Given a rooted tree, label the vertices with positive integers so that the divisor graph equals the rooted tree.

Solution

1. Put the value 1 on the root, there is no reason not to.

F — Factor-Full Tree

Problem

Given a rooted tree, label the vertices with positive integers so that the divisor graph equals the rooted tree.

Solution

1. Put the value 1 on the root, there is no reason not to.
2. If the tree is a line, then the best we can do is to use powers of 2. The largest number becomes $2^{N-1} \leq 2^{59} < 10^{18}$.

F — Factor-Full Tree

Problem

Given a rooted tree, label the vertices with positive integers so that the divisor graph equals the rooted tree.

Solution

1. Put the value 1 on the root, there is no reason not to.
2. If the tree is a line, then the best we can do is to use powers of 2. The largest number becomes $2^{N-1} \leq 2^{59} < 10^{18}$.
3. A useful reduction: instead of x_v on vertices, consider the values $y_e = x_v / x_u$ that we put on edges. These should be small primes.

F — Factor-Full Tree

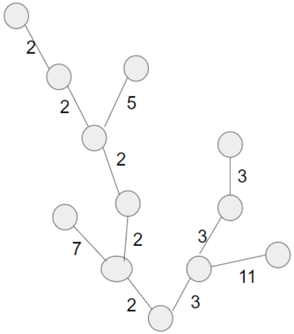
Problem

Given a rooted tree, label the vertices with positive integers so that the divisor graph equals the rooted tree.

Solution

1. Put the value 1 on the root, there is no reason not to.
2. If the tree is a line, then the best we can do is to use powers of 2. The largest number becomes $2^{N-1} \leq 2^{59} < 10^{18}$.
3. A useful reduction: instead of x_v on vertices, consider the values $y_e = x_v / x_u$ that we put on edges. These should be small primes.
4. It seems like a good idea to put 2s on the deepest leaf, then 3s on the second deepest, and so on.

F — Factor-Full Tree



F — Factor-Full Tree

Solution

1. Put the value 1 on the root, there is no reason not to.
2. If the tree is a line, then the best we can do is to use powers of 2. The largest number becomes $2^{N-1} \leq 2^{59} < 10^{18}$.
3. A useful reduction: instead of x_v on vertices, consider the values $y_e = x_v / x_u$ that we put on edges. These should be small primes.
4. It seems like a good idea to put 2s on the deepest leaf, then 3s on the second deepest, and so on.
5. This turns out to work. There are a few other ways to get numbers of size at most 2^{N-1} as well.

Statistics at 4-hour mark: 229 submissions, 45 accepted, first after HH:MM

E — Electronic Components

Problem

There are N types of components, each with f_i copies and placement time t_i. In one move you can place two components of different types i and j in $\max(t_i, t_j)$ nanoseconds. Find the minimum total time to place all components.

Solution

E — Electronic Components

Problem

There are N types of components, each with f_i copies and placement time t_i. In one move you can place two components of different types i and j in $\max(t_i, t_j)$ nanoseconds. Find the minimum total time to place all components.

Solution

1. Greedily place two component types with maximum t_i? Doesn't work on second sample.
2. If t_i values are similar, then it is more important to match all components.

E — Electronic Components

Solution

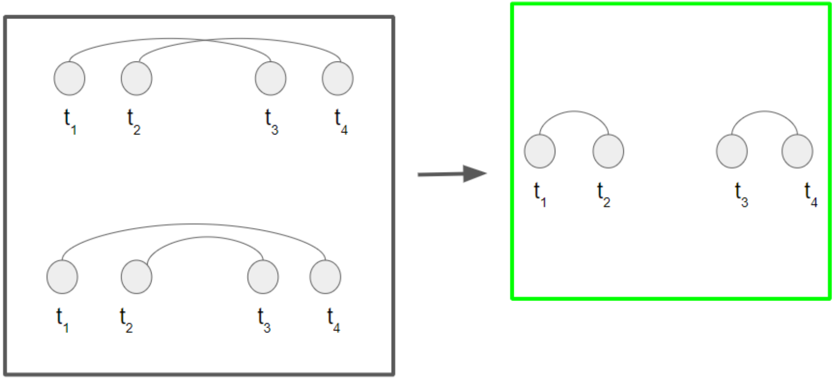
1. Let's try DP instead. Sort the types with respect to time.

E — Electronic Components

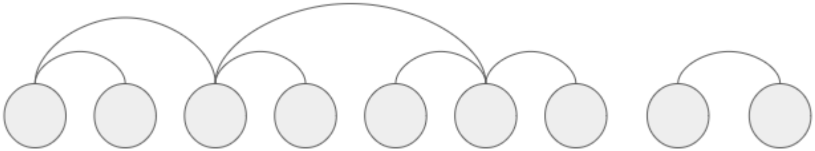
Solution

1. Let's try DP instead. Sort the types with respect to time.
2. $DP(i, u) =$ answer of first i components, if there are u unmatched components that we already "paid" for.
3. Number of states is N^2F, transitions take $\mathcal{O}(F)$. Running time $\mathcal{O}(N^2F^2)$.

E — Electronic Components



E — Electronic Components



E — Electronic Components

Solution

1. $DP(i, u) =$ answer of first i components, if there are u unmatched components that we already "paid" for.
2. Number of states is N^2F, transitions take $\mathcal{O}(F)$. Running time $\mathcal{O}(N^2F^2)$.

Statistics at 4-hour mark: 45 submissions, 2 accepted, first after HH:MM

E — Electronic Components

Solution

1. $DP(i, u) =$ answer of first i components, if there are u unmatched components that we already "paid" for.
2. Number of states is N^2F, transitions take $\mathcal{O}(F)$. Running time $\mathcal{O}(N^2F^2)$.
3. The value of u never exceeds F in an optimal solution.

Statistics at 4-hour mark: 45 submissions, 2 accepted, first after HH:MM

E — Electronic Components

Solution

1. $DP(i, u) =$ answer of first i components, if there are u unmatched components that we already "paid" for.
2. Number of states is $N^2 F$, transitions take $\mathcal{O}(F)$. Running time $\mathcal{O}(N^2 F^2)$.
3. The value of u never exceeds F in an optimal solution.
4. Write out the DP transitions carefully. They can be reduced to RMQ:s on a sliding window.

Statistics at 4-hour mark: 45 submissions, 2 accepted, first after HH:MM

E — Electronic Components

Solution

1. $DP(i, u) =$ answer of first i components, if there are u unmatched components that we already "paid" for.
2. Number of states is N^2F, transitions take $\mathcal{O}(F)$. Running time $\mathcal{O}(N^2F^2)$.
3. The value of u never exceeds F in an optimal solution.
4. Write out the DP transitions carefully. They can be reduced to RMQ:s on a sliding window.
5. Use segment tree, RMQ, or priority queue in the transition: $\mathcal{O}(NF \log(F))$.
6. Bonus: can you solve it in $\mathcal{O}(NF)$ (without fancy RMQ)?

Statistics at 4-hour mark: 45 submissions, 2 accepted, first after HH:MM

G — Groups of Strangers

Problem

Find a proper three-coloring of the $N \leq 1000$ vertices in an undirected graph on $M \leq 100000$ edges, or report that none exists.

Solution - first part

G — Groups of Strangers

Problem

Find a proper three-coloring of the $N \leq 1000$ vertices in an undirected graph on $M \leq 100000$ edges, or report that none exists.

Solution - first part

1. Three-coloring is NP-hard - we need backdoor.

G — Groups of Strangers

Problem

Find a proper three-coloring of the $N \leq 1000$ vertices in an undirected graph on $M \leq 100000$ edges, or report that none exists.

Solution - first part

1. Three-coloring is NP-hard - we need backdoor.
2. The backdoor here is a vertex that has a subset of few neighbors, say at most $B \leq 12$, through which it reaches every other vertex in at most two steps.

G — Groups of Strangers

Problem

Find a proper three-coloring of the $N \leq 1000$ vertices in an undirected graph on $M \leq 100000$ edges, or report that none exists.

Solution - first part

1. Three-coloring is NP-hard - we need backdoor.
2. The backdoor here is a vertex that has a subset of few neighbors, say at most $B \leq 12$, through which it reaches every other vertex in at most two steps.
3. Such a vertex is guaranteed to exist (e.g., the CEO even has $B \leq 8$). In case there are many, anyone will do!

G — Groups of Strangers

Problem

Find a proper three-coloring of the $N \leq 1000$ vertices in an undirected graph on $M \leq 100000$ edges, or report that none exists.

Solution - first part

1. Three-coloring is NP-hard - we need backdoor.
2. The backdoor here is a vertex that has a subset of few neighbors, say at most $B \leq 12$, through which it reaches every other vertex in at most two steps.
3. Such a vertex is guaranteed to exist (e.g., the CEO even has $B \leq 8$). In case there are many, anyone will do!
4. To find backdoor, first find candidate "CEO" vertices of degree ≤ 15 that reaches every vertex in at most two steps in $O(n^2)$ time.

G — Groups of Strangers

Problem

Find a proper three-coloring of the $N \leq 1000$ vertices in an undirected graph on $M \leq 100000$ edges, or report that none exists.

Solution - first part

1. Three-coloring is NP-hard - we need backdoor.
2. The backdoor here is a vertex that has a subset of few neighbors, say at most $B \leq 12$, through which it reaches every other vertex in at most two steps.
3. Such a vertex is guaranteed to exist (e.g., the CEO even has $B \leq 8$). In case there are many, anyone will do!
4. To find backdoor, first find candidate "CEO" vertices of degree ≤ 15 that reaches every vertex in at most two steps in $O(n^2)$ time.
5. Then for subsets of each candidate CEO's neighbors from larger to smaller, see if they still cover the graph. $O(\sum_{k=B}^{15} \binom{15}{k} n)$ time per candidate CEO.

G — Groups of Strangers

Problem

Find a proper three-coloring of the $N \leq 1000$ vertices in an undirected graph on $M \leq 100000$ edges, or report that none exists.

Solution - second part

Statistics at 4-hour mark: 18 submissions, 2 accepted, first after HH:MM

G — Groups of Strangers

Problem

Find a proper three-coloring of the $N \leq 1000$ vertices in an undirected graph on $M \leq 100000$ edges, or report that none exists.

Solution - second part

1. Pick an arbitrary color for the CEO, and test all 2^B ways to color the at most B spanning neighbors.

Statistics at 4-hour mark: 18 submissions, 2 accepted, first after HH:MM

G — Groups of Strangers

Problem

Find a proper three-coloring of the $N \leq 1000$ vertices in an undirected graph on $M \leq 100000$ edges, or report that none exists.

Solution - second part

1. Pick an arbitrary color for the CEO, and test all 2^B ways to color the at most B spanning neighbors.
2. Reduce to 2SAT since every vertex now have at most two possible colors. $O(N + M)$ time per tried coloring of spanning neighbors.

Statistics at 4-hour mark: 18 submissions, 2 accepted, first after HH:MM

I — Intertwined

Problem

You are given n points in the plane along with a rope of length d. As it wraps around the origin you want to find out which point it ends up wrapping around.

Solution

1. Imagine drawing a line through the rope and taking the convex hull of the points on the left side of the rope.

I — Intertwined

Problem

You are given n points in the plane along with a rope of length d. As it wraps around the origin you want to find out which point it ends up wrapping around.

Solution

1. Imagine drawing a line through the rope and taking the convex hull of the points on the left side of the rope.
2. Then as you rotate the line, the next point you would touch (aside from when the rope is too short) is on this hull.

I — Intertwined

Problem

You are given n points in the plane along with a rope of length d. As it wraps around the origin you want to find out which point it ends up wrapping around.

Solution

1. Imagine drawing a line through the rope and taking the convex hull of the points on the left side of the rope.
2. Then as you rotate the line, the next point you would touch (aside from when the rope is too short) is on this hull.
3. Thus we can solve this by maintaining convex hulls. When we miss a point due to the rope being too short we can delete that point instead.

I — Intertwined

Solution

1. Using a decremental convex hull data structure we can maintain the hulls in $O(n\log(n))$ with $O(\log(n))$ deletion.

Statistics at 4-hour mark: 6 submissions, 0 accepted, first after HH:MM

I — Intertwined

Solution

1. Using a decremental convex hull data structure we can maintain the hulls in $O(n\log(n))$ with $O(\log(n))$ deletion.
2. This data structure consists of two halves, each maintaining an upper and lower hull. They consist of a tree where the leaves are the hull points and when a point is deleted, the change is propagated to maintain the hull.

Statistics at 4-hour mark: 6 submissions, 0 accepted, first after HH:MM

I — Intertwined

Solution

1. Using a decremental convex hull data structure we can maintain the hulls in $O(n\log(n))$ with $O(\log(n))$ deletion.
2. This data structure consists of two halves, each maintaining an upper and lower hull. They consist of a tree where the leaves are the hull points and when a point is deleted, the change is propagated to maintain the hull.
3. We maintain one data structure for the upper half plane and iterate until we touch the hull for all points, then move to maintaining the data structure for all points instead.

Statistics at 4-hour mark: 6 submissions, 0 accepted, first after HH:MM

I — Intertwined

Solution

1. Using a decremental convex hull data structure we can maintain the hulls in $O(n \log(n))$ with $O(\log(n))$ deletion.
2. This data structure consists of two halves, each maintaining an upper and lower hull. They consist of a tree where the leaves are the hull points and when a point is deleted, the change is propagated to maintain the hull.
3. We maintain one data structure for the upper half plane and iterate until we touch the hull for all points, then move to maintaining the data structure for all points instead.
4. Edge case: If we travel around the entire hull without deleting any points we have to modulo the rope length with the distance traveled so we don't go in $\approx 10^9$ circles.

Statistics at 4-hour mark: 6 submissions, 0 accepted, first after HH:MM

B — Berry Battle 2

Problem

There are $N/2$ berries randomly placed in a sequence of length N. You play a game where the players take turn picking all berries in 4 adjacent positions. Your opponent makes the first move, and follows a simple greedy strategy. Your goal is to get at least as many berries.

B — Berry Battle 2

Problem

There are $N/2$ berries randomly placed in a sequence of length N. You play a game where the players take turn picking all berries in 4 adjacent positions. Your opponent makes the first move, and follows a simple greedy strategy. Your goal is to get at least as many berries.

Solution

1. It is impossible to pick more berries than grandpa.

B — Berry Battle 2

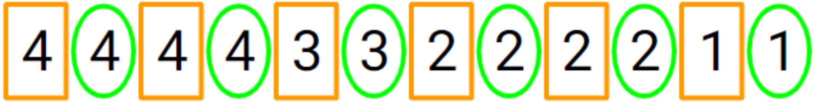
Problem

There are $N/2$ berries randomly placed in a sequence of length N. You play a game where the players take turn picking all berries in 4 adjacent positions. Your opponent makes the first move, and follows a simple greedy strategy. Your goal is to get at least as many berries.

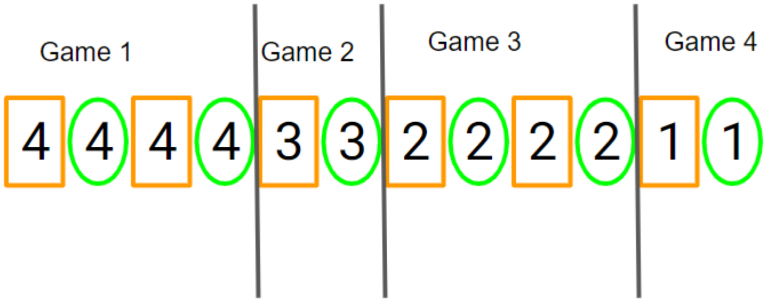
Solution

1. It is impossible to pick more berries than grandpa.
2. The only way to tie the game is if our scores mirror grandpa's scores.

B — Berry Battle 2



B — Berry Battle 2



B — Berry Battle 2

1. The game consists of four phases. In each phase, we want to make the last move.

B — Berry Battle 2

1. The game consists of four phases. In each phase, we want to make the last move.
2. **Main strategy:** Each phase is an impartial game. We will play each game perfectly by calculating Grundy numbers.

B — Berry Battle 2

Why and how

1. Grandpa plays the four impartial games randomly. Each game consists of a few thousand moves. If he has a winning strategy, then it is very likely that he will make a mistake.

Statistics at 4-hour mark: 17 submissions, 0 accepted, first after HH:MM

B — Berry Battle 2

Why and how

1. Grandpa plays the four impartial games randomly. Each game consists of a few thousand moves. If he has a winning strategy, then it is very likely that he will make a mistake.
2. The Grundy numbers can be calculated in $\mathcal{O}(N^3)$ with DP, which is too slow. We need various tricks to speed it up.

Statistics at 4-hour mark: 17 submissions, 0 accepted, first after HH:MM

B — Berry Battle 2

Why and how

1. Grandpa plays the four impartial games randomly. Each game consists of a few thousand moves. If he has a winning strategy, then it is very likely that he will make a mistake.
2. The Grundy numbers can be calculated in $\mathcal{O}(N^3)$ with DP, which is too slow. We need various tricks to speed it up.
3. **Main speedup:** The sequence has many gaps with no berries, so the game can be partitioned into many small independent games. Find the Grundy number of each and XOR them.

Statistics at 4-hour mark: 17 submissions, 0 accepted, first after HH:MM

Results!