NCPC 2021
Presentation of solutions

2021-10-09

Problems prepared by

- Per Austrin (KTH Royal Institute of Technology)
- Andreas Björklund (Apptus)
- Pål Grønås Drange (University of Bergen)
- Nils Gustafsson (KTH Royal Institute of Technology)
- Antti Laaksonen (CSES)
- Jimmy Mårdell (Spotify)
- Bergur Snorrason (University of Iceland)

K — Knot Knowledge

Problem

You are given a list of numbers and then another list of the same numbers, except one number has been removed. What number was removed?

Solution

K — Knot Knowledge

Problem

You are given a list of numbers and then another list of the same numbers, except one number has been removed. What number was removed?

Solution

1. This problem can be solved in many ways, the list is small so there is no need for any special data structure or algorithm.

K — Knot Knowledge

Problem

You are given a list of numbers and then another list of the same numbers, except one number has been removed. What number was removed?

Solution

1. This problem can be solved in many ways, the list is small so there is no need for any special data structure or algorithm.
2. But for this problem, the most efficient solution is also the simplest:

K — Knot Knowledge

Problem

You are given a list of numbers and then another list of the same numbers, except one number has been removed. What number was removed?

Solution

1. This problem can be solved in many ways, the list is small so there is no need for any special data structure or algorithm.
2. But for this problem, the most efficient solution is also the simplest:
3. Let the first list of numbers be $x_1, x_2, \ldots x_n$ and the second list be $y_1, y_2 \ldots y_{n-1}$.

K — Knot Knowledge

Problem

You are given a list of numbers and then another list of the same numbers, except one number has been removed. What number was removed?

Solution

1. This problem can be solved in many ways, the list is small so there is no need for any special data structure or algorithm.
2. But for this problem, the most efficient solution is also the simplest:
3. Let the first list of numbers be $x_1, x_2, \ldots x_n$ and the second list be $y_1, y_2 \ldots y_{n-1}$.
4. The answer is then

$$\sum_{k=1}^{n} x_k - \sum_{k=1}^{n-1} y_k.$$

K — Knot Knowledge

Problem

You are given a list of numbers and then another list of the same numbers, except one number has been removed. What number was removed?

Solution

1. This problem can be solved in many ways, the list is small so there is no need for any special data structure or algorithm.
2. But for this problem, the most efficient solution is also the simplest:
3. Let the first list of numbers be $x_1, x_2, \ldots x_n$ and the second list be $y_1, y_2 \ldots y_{n-1}$.
4. The answer is then

$$\sum_{k=1}^{n} x_k - \sum_{k=1}^{n-1} y_k.$$

Statistics at 4-hour mark: 248 submissions, 196 accepted, first after 00:01

J — Joint Jog Jam

Problem

Compute the maximum distance between two people moving in straight line segments at constant speed.

Solution

J — Joint Jog Jam

Problem

Compute the maximum distance between two people moving in straight line segments at constant speed.

Solution

1. The answer is simply the max of the distance between the starting points and the distance between the ending points. Let us see why...

J — Joint Jog Jam

Problem

Compute the maximum distance between two people moving in straight line segments at constant speed.

Solution

1. The answer is simply the max of the distance between the starting points and the distance between the ending points. Let us see why...
2. Useful trick: by viewing everything from the reference frame of the first runner, we can instead assume that the first person is stationary at $(0, 0)$.

J — Joint Jog Jam

Problem

Compute the maximum distance between two people moving in straight line segments at constant speed.

Solution

1. The answer is simply the max of the distance between the starting points and the distance between the ending points. Let us see why...
2. Useful trick: by viewing everything from the reference frame of the first runner, we can instead assume that the first person is stationary at $(0, 0)$.
3. Second person is at some position $(x(t), y(t)) = (x_0 + t \cdot x_\delta, y_0 + t \cdot y_\delta)$ at time t.

J — Joint Jog Jam

Problem

Compute the maximum distance between two people moving in straight line segments at constant speed.

Solution

1. The answer is simply the max of the distance between the starting points and the distance between the ending points. Let us see why...
2. Useful trick: by viewing everything from the reference frame of the first runner, we can instead assume that the first person is stationary at $(0, 0)$.
3. Second person is at some position $(x(t), y(t)) = (x_0 + t \cdot x_\delta, y_0 + t \cdot y_\delta)$ at time t.
4. Squared distance at time t is then $x(t)^2 + y(t)^2 = (x_0 + t \cdot x_\delta)^2 + (y_0 + t \cdot y_\delta)^2$

J — Joint Jog Jam

Problem

Compute the maximum distance between two people moving in straight line segments at constant speed.

Solution

1. The answer is simply the max of the distance between the starting points and the distance between the ending points. Let us see why...
2. Useful trick: by viewing everything from the reference frame of the first runner, we can instead assume that the first person is stationary at $(0, 0)$.
3. Second person is at some position $(x(t), y(t)) = (x_0 + t \cdot x_\delta, y_0 + t \cdot y_\delta)$ at time t.
4. Squared distance at time t is then $x(t)^2 + y(t)^2 = (x_0 + t \cdot x_\delta)^2 + (y_0 + t \cdot y_\delta)^2$
5. This is a convex quadratic function in t, so it is maximized at $t = t_{\text{min}}$ or $t = t_{\text{max}}$.

J — Joint Jog Jam

Problem

Compute the maximum distance between two people moving in straight line segments at constant speed.

Solution

1. The answer is simply the max of the distance between the starting points and the distance between the ending points. Let us see why...
2. Useful trick: by viewing everything from the reference frame of the first runner, we can instead assume that the first person is stationary at $(0, 0)$.
3. Second person is at some position $(x(t), y(t)) = (x_0 + t \cdot x_\delta, y_0 + t \cdot y_\delta)$ at time t.
4. Squared distance at time t is then $x(t)^2 + y(t)^2 = (x_0 + t \cdot x_\delta)^2 + (y_0 + t \cdot y_\delta)^2$
5. This is a convex quadratic function in t, so it is maximized at $t = t_{\text{min}}$ or $t = t_{\text{max}}$.

Statistics at 4-hour mark: 274 submissions, 189 accepted, first after 00:10

L — Locust Locus

Problem

A pair of cicada species appear every a and b years, and was last observed in year y. Find the next year they will appear together again.

L — Locust Locus

Problem

A pair of cicada species appear every a and b years, and was last observed in year y. Find the next year they will appear together again.

Solution 1

1. For each year $z = y + 1, y + 2, y + 3, \ldots$:
   - Check if $z - y$ is divisible by both a and b.
   - Break when first such z is found.

L — Locust Locus

Problem

A pair of cicada species appear every a and b years, and was last observed in year y. Find the next year they will appear together again.

Solution 1

1. For each year $z = y + 1, y + 2, y + 3, \ldots$:
    - Check if $z - y$ is divisible by both a and b.
    - Break when first such z is found.
2. Goes on for at most $a \cdot b$ steps because $a \cdot b$ definitely divides both a and b.

L — Locust Locus

Problem

A pair of cicada species appear every a and b years, and was last observed in year y. Find the next year they will appear together again.

Solution 1

1. For each year $z = y + 1, y + 2, y + 3, \ldots$:
   - Check if $z - y$ is divisible by both a and b.
   - Break when first such z is found.
2. Goes on for at most $a \cdot b$ steps because $a \cdot b$ definitely divides both a and b.
3. So this takes $O(a \cdot b)$ time which is fast enough because a and b are very small.

L — Locust Locus

Problem

A pair of cicada species appear every a and b years, and was last observed in year y. Find the next year they will appear together again.

Solution 2

1. The joint period of the two species is the least common multiple of a and b.

L — Locust Locus

Problem

A pair of cicada species appear every a and b years, and was last observed in year y. Find the next year they will appear together again.

Solution 2

1. The joint period of the two species is the least common multiple of a and b.
2. Answer is $y + \mathrm{lcm}(a, b)$.

L — Locust Locus

Problem

A pair of cicada species appear every a and b years, and was last observed in year y. Find the next year they will appear together again.

Solution 2

1. The joint period of the two species is the least common multiple of a and b.
2. Answer is $y + \operatorname{lcm}(a, b)$.
3. Takes $O(\log(a \cdot b))$ time to compute.

L — Locust Locus

Problem

A pair of cicada species appear every a and b years, and was last observed in year y. Find the next year they will appear together again.

Solution 2

1. The joint period of the two species is the least common multiple of a and b.
2. Answer is $y + \operatorname{lcm}(a, b)$.
3. Takes $O(\log(a \cdot b))$ time to compute.

Statistics at 4-hour mark: 374 submissions, 185 accepted, first after 00:02

G — Grazed Grains

Problem

Given $n \leq 10$ circles of radius ≤ 10 and with centers in $[0, 10] \times [0, 10]$, approximate the area of their union, up to a factor 1 ± 0.1.

Solutiom

G — Grazed Grains

Problem

Given $n \leq 10$ circles of radius ≤ 10 and with centers in $[0, 10] \times [0, 10]$, approximate the area of their union, up to a factor 1 ± 0.1.

Solutiom

1. Can compute the area with high precision using numeric integration.
   Not too hard, but a bit of code, and there is a simpler solution: use sampling.

G — Grazed Grains

Problem

Given $n \leq 10$ circles of radius ≤ 10 and with centers in $[0, 10] \times [0, 10]$, approximate the area of their union, up to a factor 1 ± 0.1.

Solutiom

1. Can compute the area with high precision using numeric integration.
   Not too hard, but a bit of code, and there is a simpler solution: use sampling.
2. Sample r uniformly random points in $[-10, 20] \times [-10, 20]$.
   (this box chosen so that all the circles are contained in it)

G — Grazed Grains

Problem

Given $n \leq 10$ circles of radius ≤ 10 and with centers in $[0, 10] \times [0, 10]$, approximate the area of their union, up to a factor 1 ± 0.1.

Solutiom

1. Can compute the area with high precision using numeric integration. Not too hard, but a bit of code, and there is a simpler solution: use sampling.
2. Sample r uniformly random points in $[-10, 20] \times [-10, 20]$. (this box chosen so that all the circles are contained in it)
3. If x of the r points are inside some circle, we estimate the area as $\frac{x}{r} \cdot 30^2$.

G — Grazed Grains

Problem

Given $n \leq 10$ circles of radius ≤ 10 and with centers in $[0, 10] \times [0, 10]$, approximate the area of their union, up to a factor 1 ± 0.1.

Solutiom

1. Can compute the area with high precision using numeric integration. Not too hard, but a bit of code, and there is a simpler solution: use sampling.
2. Sample r uniformly random points in $[-10, 20] \times [-10, 20]$. (this box chosen so that all the circles are contained in it)
3. If x of the r points are inside some circle, we estimate the area as $\frac{x}{r} \cdot 30^2$.
4. Analysis (not needed to solve the problem): can prove that you expect a relative error around $24/\sqrt{r}$. If $r > 100$k this starts becoming small enough, and with $r = 1$ million the sampling error is very unlikely to be too large.

G — Grazed Grains

Problem

Given $n \leq 10$ circles of radius ≤ 10 and with centers in $[0, 10] \times [0, 10]$, approximate the area of their union, up to a factor 1 ± 0.1.

Solutiom

1. Can compute the area with high precision using numeric integration.
   Not too hard, but a bit of code, and there is a simpler solution: use sampling.
2. Sample r uniformly random points in $[-10, 20] \times [-10, 20]$.
   (this box chosen so that all the circles are contained in it)
3. If x of the r points are inside some circle, we estimate the area as $\frac{x}{r} \cdot 30^2$.
4. Analysis (not needed to solve the problem): can prove that you expect a relative error around $24/\sqrt{r}$. If $r > 100$k this starts becoming small enough, and with $r = 1$ million the sampling error is very unlikely to be too large.

Statistics at 4-hour mark: 267 submissions, 117 accepted, first after 00:07

A — Antenna Analysis

Problem

Given integers $x_1, \ldots, x_n$, find, for each i, the maximum of $|x_i - x_j| - c|i - j|$ over $j \leq i$.

Solution

A — Antenna Analysis

Problem

Given integers $x_1, \ldots, x_n$, find, for each i, the maximum of $|x_i - x_j| - c|i - j|$ over $j \leq i$.

Solution

1. Key observation: dropping the absolute values does not increase the answer.

A — Antenna Analysis

Problem

Given integers $x_1, \ldots, x_n$, find, for each i, the maximum of $|x_i - x_j| - c|i - j|$ over $j \leq i$.

Solution

1. Key observation: dropping the absolute values does not increase the answer.
2. Simplify problem: drop the absolute values and maximize $(x_i - x_j) - c(i - j) = (x_i - c \cdot i) + (-x_j + c \cdot j)$.

A — Antenna Analysis

Problem

Given integers $x_1, \ldots, x_n$, find, for each i, the maximum of $|x_i - x_j| - c|i - j|$ over $j \le i$.

Solution

1. Key observation: dropping the absolute values does not increase the answer.
2. Simplify problem: drop the absolute values and maximize $(x_i - x_j) - c(i - j) = (x_i - c \cdot i) + (-x_j + c \cdot j)$.
   - Keep track of largest value y of $-x_j + c \cdot j$ seen so far.

A — Antenna Analysis

Problem

Given integers $x_1, \ldots, x_n$, find, for each i, the maximum of $|x_i - x_j| - c|i - j|$ over $j \leq i$.

Solution

1. Key observation: dropping the absolute values does not increase the answer.
2. Simplify problem: drop the absolute values and maximize $(x_i - x_j) - c(i - j) = (x_i - c \cdot i) + (-x_j + c \cdot j)$.
   - Keep track of largest value y of $-x_j + c \cdot j$ seen so far.
   - Answer to simplified problem is $x_i - c \cdot i + y$.

A — Antenna Analysis

Problem

Given integers $x_1, \ldots, x_n$, find, for each i, the maximum of $|x_i - x_j| - c|i - j|$ over $j \leq i$.

Solution

1. Key observation: dropping the absolute values does not increase the answer.
2. Simplify problem: drop the absolute values and maximize $(x_i - x_j) - c(i - j) = (x_i - c \cdot i) + (-x_j + c \cdot j)$.
   - Keep track of largest value y of $-x_j + c \cdot j$ seen so far.
   - Answer to simplified problem is $x_i - c \cdot i + y$.
3. To solve original problem, separately solve variant for $(x_j - x_i) - c(i - j)$ the same way and take largest of the two.

A — Antenna Analysis

Problem

Given integers $x_1, \ldots, x_n$, find, for each i, the maximum of $|x_i - x_j| - c|i - j|$ over $j \leq i$.

Solution

1. Key observation: dropping the absolute values does not increase the answer.
2. Simplify problem: drop the absolute values and maximize $(x_i - x_j) - c(i - j) = (x_i - c \cdot i) + (-x_j + c \cdot j)$.
   - Keep track of largest value y of $-x_j + c \cdot j$ seen so far.
   - Answer to simplified problem is $x_i - c \cdot i + y$.
3. To solve original problem, separately solve variant for $(x_j - x_i) - c(i - j)$ the same way and take largest of the two.
4. Time complexity $O(n)$.

A — Antenna Analysis

Problem

Given integers $x_1, \ldots, x_n$, find, for each i, the maximum of $|x_i - x_j| - c|i - j|$ over $j \leq i$.

Solution

1. Key observation: dropping the absolute values does not increase the answer.
2. Simplify problem: drop the absolute values and maximize $(x_i - x_j) - c(i - j) = (x_i - c \cdot i) + (-x_j + c \cdot j)$.
   - Keep track of largest value y of $-x_j + c \cdot j$ seen so far.
   - Answer to simplified problem is $x_i - c \cdot i + y$.
3. To solve original problem, separately solve variant for $(x_j - x_i) - c(i - j)$ the same way and take largest of the two.
4. Time complexity $O(n)$.
5. More complicated solutions using balanced search trees or range trees also possible.

A — Antenna Analysis

Problem

Given integers $x_1, \ldots, x_n$, find, for each i, the maximum of $|x_i - x_j| - c|i - j|$ over $j \leq i$.

Solution

1. Key observation: dropping the absolute values does not increase the answer.
2. Simplify problem: drop the absolute values and maximize $(x_i - x_j) - c(i - j) = (x_i - c \cdot i) + (-x_j + c \cdot j)$.
   - Keep track of largest value y of $-x_j + c \cdot j$ seen so far.
   - Answer to simplified problem is $x_i - c \cdot i + y$.
3. To solve original problem, separately solve variant for $(x_j - x_i) - c(i - j)$ the same way and take largest of the two.
4. Time complexity $O(n)$.
5. More complicated solutions using balanced search trees or range trees also possible.

Statistics at 4-hour mark: 626 submissions, 88 accepted, first after 00:04

D — Deceptive Directions

Problem

Get $w \times h$ grid map and a shortest sequence of NWSE steps to reach some treasure. But all the steps have been replaced by wrong ones. Where could the treasure be?

Solution

D — Deceptive Directions

Problem

Get $w \times h$ grid map and a shortest sequence of NWSE steps to reach some treasure. But all the steps have been replaced by wrong ones. Where could the treasure be?

Solution

1. Compute distance $d[p]$ from start to every position p in the maze using BFS.

D — Deceptive Directions

Problem

Get $w \times h$ grid map and a shortest sequence of NWSE steps to reach some treasure. But all the steps have been replaced by wrong ones. Where could the treasure be?

Solution

1. Compute distance $d[p]$ from start to every position p in the maze using BFS.
2. Forbid all steps that could not have been used by the correct instruction sequence:

D — Deceptive Directions

Problem

Get $w \times h$ grid map and a shortest sequence of NWSE steps to reach some treasure. But all the steps have been replaced by wrong ones. Where could the treasure be?

Solution

1. Compute distance $d[p]$ from start to every position p in the maze using BFS.
2. Forbid all steps that could not have been used by the correct instruction sequence:
   - Forbid step $p \to q$ if not used in any shortest path (i.e., if $d[q] \neq d[p] + 1$).

D — Deceptive Directions

Problem

Get $w \times h$ grid map and a shortest sequence of NWSE steps to reach some treasure. But all the steps have been replaced by wrong ones. Where could the treasure be?

Solution

1. Compute distance $d[p]$ from start to every position p in the maze using BFS.
2. Forbid all steps that could not have been used by the correct instruction sequence:
   - Forbid step $p \to q$ if not used in any shortest path (i.e., if $d[q] \neq d[p] + 1$).
   - Forbid step $p \to q$ if going from p to q means taking the same step as the $d[q]$'th step in the corrupted instruction sequence.

D — Deceptive Directions

Problem

Get $w \times h$ grid map and a shortest sequence of NWSE steps to reach some treasure. But all the steps have been replaced by wrong ones. Where could the treasure be?

Solution

1. Compute distance $d[p]$ from start to every position p in the maze using BFS.
2. Forbid all steps that could not have been used by the correct instruction sequence:
   - Forbid step $p \to q$ if not used in any shortest path (i.e., if $d[q] \neq d[p] + 1$).
   - Forbid step $p \to q$ if going from p to q means taking the same step as the $d[q]$'th step in the corrupted instruction sequence.
3. Run BFS again with these steps forbidden.

D — Deceptive Directions

Problem

Get $w \times h$ grid map and a shortest sequence of NWSE steps to reach some treasure. But all the steps have been replaced by wrong ones. Where could the treasure be?

Solution

1. Compute distance $d[p]$ from start to every position p in the maze using BFS.
2. Forbid all steps that could not have been used by the correct instruction sequence:
   - Forbid step $p \to q$ if not used in any shortest path (i.e., if $d[q] \neq d[p] + 1$).
   - Forbid step $p \to q$ if going from p to q means taking the same step as the $d[q]$'th step in the corrupted instruction sequence.
3. Run BFS again with these steps forbidden.
   - All reached positions at same distance as instruction length are possible treasure locations.

D — Deceptive Directions

Problem

Get $w \times h$ grid map and a shortest sequence of NWSE steps to reach some treasure. But all the steps have been replaced by wrong ones. Where could the treasure be?

Solution

1. Compute distance $d[p]$ from start to every position p in the maze using BFS.
2. Forbid all steps that could not have been used by the correct instruction sequence:
   - Forbid step $p \to q$ if not used in any shortest path (i.e., if $d[q] \neq d[p] + 1$).
   - Forbid step $p \to q$ if going from p to q means taking the same step as the $d[q]$'th step in the corrupted instruction sequence.
3. Run BFS again with these steps forbidden.
   - All reached positions at same distance as instruction length are possible treasure locations.
4. Time complexity: $O(w \cdot h)$.

D — Deceptive Directions

Problem

Get $w \times h$ grid map and a shortest sequence of NWSE steps to reach some treasure. But all the steps have been replaced by wrong ones. Where could the treasure be?

Solution

1. Compute distance $d[p]$ from start to every position p in the maze using BFS.
2. Forbid all steps that could not have been used by the correct instruction sequence:
   - Forbid step $p \to q$ if not used in any shortest path (i.e., if $d[q] \neq d[p] + 1$).
   - Forbid step $p \to q$ if going from p to q means taking the same step as the $d[q]$'th step in the corrupted instruction sequence.
3. Run BFS again with these steps forbidden.
   - All reached positions at same distance as instruction length are possible treasure locations.
4. Time complexity: $O(w \cdot h)$.

Statistics at 4-hour mark: 325 submissions, 55 accepted, first after 00:38

F — Fortune From Folly

Problem

In infinite random binary sequence $x_1, x_2, x_3, \ldots$ where each $x_i = 1$ with probability p (independently), what is expected first value of i such that $x_i + x_{i-1} + \ldots + x_{i-n+1} \geq k$?

Solution

F — Fortune From Folly

Problem

In infinite random binary sequence $x_1, x_2, x_3, \ldots$ where each $x_i = 1$ with probability p (independently), what is expected first value of i such that $x_i + x_{i-1} + \ldots + x_{i-n+1} \geq k$?

Solution

1. At any point, only the n most recent x_i's matter.

F — Fortune From Folly

Problem

In infinite random binary sequence $x_1, x_2, x_3, \ldots$ where each $x_i = 1$ with probability p (independently), what is expected first value of i such that $x_i + x_{i-1} + \ldots + x_{i-n+1} \geq k$?

Solution

1. At any point, only the n most recent x_i's matter.
2. Let $E_{z_1 z_2 z_3 \ldots z_n}$ be expected #steps until k ones, if most recent x_i's are $z_1, \ldots, z_n$.

F — Fortune From Folly

Problem

In infinite random binary sequence $x_1, x_2, x_3, \ldots$ where each $x_i = 1$ with probability p (independently), what is expected first value of i such that $x_i + x_{i-1} + \ldots + x_{i-n+1} \geq k$?

Solution

1. At any point, only the n most recent x_i's matter.
2. Let $E_{z_1 z_2 z_3 \ldots z_n}$ be expected #steps until k ones, if most recent x_i's are $z_1, \ldots, z_n$.
   - If $\sum_{i=1}^{n} z_i \geq k$ then $E_{z_1 z_2 z_3 \ldots z_n} = 0$.

F — Fortune From Folly

Problem

In infinite random binary sequence $x_1, x_2, x_3, \ldots$ where each $x_i = 1$ with probability p (independently), what is expected first value of i such that $x_i + x_{i-1} + \ldots + x_{i-n+1} \geq k$?

Solution

1. At any point, only the n most recent x_i's matter.
2. Let $E_{z_1 z_2 z_3 \ldots z_n}$ be expected #steps until k ones, if most recent x_i's are $z_1, \ldots, z_n$.
   - If $\sum_{i=1}^{n} z_i \geq k$ then $E_{z_1 z_2 z_3 \ldots z_n} = 0$.
   - Otherwise $E_{z_1 z_2 z_3 \ldots z_n} = 1 + p \cdot E_{z_2 z_3 \ldots z_n 1} + (1 - p) \cdot E_{z_2 z_3 \ldots z_n 0}$.

F — Fortune From Folly

Problem

In infinite random binary sequence $x_1, x_2, x_3, \ldots$ where each $x_i = 1$ with probability p (independently), what is expected first value of i such that $x_i + x_{i-1} + \ldots + x_{i-n+1} \geq k$?

Solution

1. At any point, only the n most recent x_i's matter.
2. Let $E_{z_1 z_2 z_3 \ldots z_n}$ be expected #steps until k ones, if most recent x_i's are $z_1, \ldots, z_n$.
   - If $\sum_{i=1}^{n} z_i \geq k$ then $E_{z_1 z_2 z_3 \ldots z_n} = 0$.
   - Otherwise $E_{z_1 z_2 z_3 \ldots z_n} = 1 + p \cdot E_{z_2 z_3 \ldots z_n 1} + (1-p) \cdot E_{z_2 z_3 \ldots z_n 0}$.
3. This is a system of linear equations in the 2^n unknowns $E_{z_1 z_2 z_3 \ldots z_n}$.

F — Fortune From Folly

Problem

In infinite random binary sequence $x_1, x_2, x_3, \ldots$ where each $x_i = 1$ with probability p (independently), what is expected first value of i such that $x_i + x_{i-1} + \ldots + x_{i-n+1} \geq k$?

Solution

1. At any point, only the n most recent x_i's matter.
2. Let $E_{z_1 z_2 z_3 \ldots z_n}$ be expected #steps until k ones, if most recent x_i's are $z_1, \ldots, z_n$.
   - If $\sum_{i=1}^{n} z_i \geq k$ then $E_{z_1 z_2 z_3 \ldots z_n} = 0$.
   - Otherwise $E_{z_1 z_2 z_3 \ldots z_n} = 1 + p \cdot E_{z_2 z_3 \ldots z_n 1} + (1-p) \cdot E_{z_2 z_3 \ldots z_n 0}$.
3. This is a system of linear equations in the 2^n unknowns $E_{z_1 z_2 z_3 \ldots z_n}$.
   - Solve using Gaussian elimination to find our answer $E_{00 \ldots 0}$.

F — Fortune From Folly

Problem

In infinite random binary sequence $x_1, x_2, x_3, \ldots$ where each $x_i = 1$ with probability p (independently), what is expected first value of i such that $x_i + x_{i-1} + \ldots + x_{i-n+1} \geq k$?

Solution

1. At any point, only the n most recent x_i's matter.
2. Let $E_{z_1 z_2 z_3 \ldots z_n}$ be expected #steps until k ones, if most recent x_i's are $z_1, \ldots, z_n$.
   - If $\sum_{i=1}^{n} z_i \geq k$ then $E_{z_1 z_2 z_3 \ldots z_n} = 0$.
   - Otherwise $E_{z_1 z_2 z_3 \ldots z_n} = 1 + p \cdot E_{z_2 z_3 \ldots z_n 1} + (1-p) \cdot E_{z_2 z_3 \ldots z_n 0}$.
3. This is a system of linear equations in the 2^n unknowns $E_{z_1 z_2 z_3 \ldots z_n}$.
   - Solve using Gaussian elimination to find our answer $E_{00\ldots0}$.
   - Time complexity $O(2^{3n})$.

F — Fortune From Folly

Problem

In infinite random binary sequence $x_1, x_2, x_3, \ldots$ where each $x_i = 1$ with probability p (independently), what is expected first value of i such that $x_i + x_{i-1} + \ldots + x_{i-n+1} \geq k$?

Solution

1. At any point, only the n most recent x_i's matter.
2. Let $E_{z_1 z_2 z_3 \ldots z_n}$ be expected #steps until k ones, if most recent x_i's are $z_1, \ldots, z_n$.
   - If $\sum_{i=1}^{n} z_i \geq k$ then $E_{z_1 z_2 z_3 \ldots z_n} = 0$.
   - Otherwise $E_{z_1 z_2 z_3 \ldots z_n} = 1 + p \cdot E_{z_2 z_3 \ldots z_n 1} + (1-p) \cdot E_{z_2 z_3 \ldots z_n 0}$.
3. This is a system of linear equations in the 2^n unknowns $E_{z_1 z_2 z_3 \ldots z_n}$.
   - Solve using Gaussian elimination to find our answer $E_{00\ldots 0}$.
   - Time complexity $O(2^{3n})$.
4. Implementation note: represent the bit string $z_1 z_2 \ldots z_n$ as an n-bit number Z

F — Fortune From Folly

Problem

In infinite random binary sequence $x_1, x_2, x_3, \ldots$ where each $x_i = 1$ with probability p (independently), what is expected first value of i such that $x_i + x_{i-1} + \ldots + x_{i-n+1} \geq k$?

Solution

1. At any point, only the n most recent x_i's matter.
2. Let $E_{z_1 z_2 z_3 \ldots z_n}$ be expected #steps until k ones, if most recent x_i's are $z_1, \ldots, z_n$.
   - If $\sum_{i=1}^{n} z_i \geq k$ then $E_{z_1 z_2 z_3 \ldots z_n} = 0$.
   - Otherwise $E_{z_1 z_2 z_3 \ldots z_n} = 1 + p \cdot E_{z_2 z_3 \ldots z_n 1} + (1 - p) \cdot E_{z_2 z_3 \ldots z_n 0}$.
3. This is a system of linear equations in the 2^n unknowns $E_{z_1 z_2 z_3 \ldots z_n}$.
   - Solve using Gaussian elimination to find our answer $E_{00 \ldots 0}$.
   - Time complexity $O(2^{3n})$.
4. Implementation note: represent the bit string $z_1 z_2 \ldots z_n$ as an n-bit number Z
   - $z_2 z_3 \ldots z_n 0 \quad \longleftrightarrow \quad (\texttt{Z} >> 1)\ \texttt{OR}\ (\texttt{b} << (\texttt{n} - 1))$

F — Fortune From Folly

Problem

In infinite random binary sequence $x_1, x_2, x_3, \ldots$ where each $x_i = 1$ with probability p (independently), what is expected first value of i such that $x_i + x_{i-1} + \ldots + x_{i-n+1} \geq k$?

Solution

1. At any point, only the n most recent x_i's matter.
2. Let $E_{z_1 z_2 z_3 \ldots z_n}$ be expected #steps until k ones, if most recent x_i's are $z_1, \ldots, z_n$.
   - If $\sum_{i=1}^{n} z_i \geq k$ then $E_{z_1 z_2 z_3 \ldots z_n} = 0$.
   - Otherwise $E_{z_1 z_2 z_3 \ldots z_n} = 1 + p \cdot E_{z_2 z_3 \ldots z_n 1} + (1-p) \cdot E_{z_2 z_3 \ldots z_n 0}$.
3. This is a system of linear equations in the 2^n unknowns $E_{z_1 z_2 z_3 \ldots z_n}$.
   - Solve using Gaussian elimination to find our answer $E_{00\ldots0}$.
   - Time complexity $O(2^{3n})$.
4. Implementation note: represent the bit string $z_1 z_2 \ldots z_n$ as an n-bit number Z
   - $z_2 z_3 \ldots z_n 0 \quad \longleftrightarrow \quad \texttt{(Z >> 1) OR (b << (n - 1))}$

Statistics at 4-hour mark: 57 submissions, 27 accepted, first after 00:31

C — Customs Controls

Problem

Given a vertex-weighted graph, color k vertices red and $n - k$ vertices blue such that every shortest path from 1 to n has a monochromatic edge.

Solution

C — Customs Controls

Problem

Given a vertex-weighted graph, color k vertices red and $n - k$ vertices blue such that every shortest path from 1 to n has a monochromatic edge.

Solution

1. Dijkstra's algorithm finds all edges that are part of some shortest path from 1 to n.

C — Customs Controls

Problem

Given a vertex-weighted graph, color k vertices red and $n - k$ vertices blue such that every shortest path from 1 to n has a monochromatic edge.

Solution

1. Dijkstra's algorithm finds all edges that are part of some shortest path from 1 to n.
2. These edges form a directed acyclic graph. Find a topological ordering.

C — Customs Controls

Problem

Given a vertex-weighted graph, color k vertices red and $n - k$ vertices blue such that every shortest path from 1 to n has a monochromatic edge.

Solution

1. Dijkstra's algorithm finds all edges that are part of some shortest path from 1 to n.
2. These edges form a directed acyclic graph. Find a topological ordering.
3. Color the first k vertices in the ordering red, and the remaining ones blue:
   - A shortest path from 1 to n now only switches between red and blue once, so every shortest path on 3 or more vertices must have a monochromatic edge.

C — Customs Controls

Problem

Given a vertex-weighted graph, color k vertices red and $n - k$ vertices blue such that every shortest path from 1 to n has a monochromatic edge.

Solution

1. Dijkstra's algorithm finds all edges that are part of some shortest path from 1 to n.
2. These edges form a directed acyclic graph. Find a topological ordering.
3. Color the first k vertices in the ordering red, and the remaining ones blue:
   - A shortest path from 1 to n now only switches between red and blue once, so every shortest path on 3 or more vertices must have a monochromatic edge.
4. Special case: this does not work if there is a direct edge from 1 to n.

C — Customs Controls

Problem

Given a vertex-weighted graph, color k vertices red and $n - k$ vertices blue such that every shortest path from 1 to n has a monochromatic edge.

Solution

1. Dijkstra's algorithm finds all edges that are part of some shortest path from 1 to n.
2. These edges form a directed acyclic graph. Find a topological ordering.
3. Color the first k vertices in the ordering red, and the remaining ones blue:
   - A shortest path from 1 to n now only switches between red and blue once, so every shortest path on 3 or more vertices must have a monochromatic edge.
4. Special case: this does not work if there is a direct edge from 1 to n.
   - Since graph is vertex-weighted, edge from 1 to n is the *only* shortest path.
   - We only need to make sure 1 and n get the same color.

C — Customs Controls

Problem

Given a vertex-weighted graph, color k vertices red and $n - k$ vertices blue such that every shortest path from 1 to n has a monochromatic edge.

Solution

1. Dijkstra's algorithm finds all edges that are part of some shortest path from 1 to n.
2. These edges form a directed acyclic graph. Find a topological ordering.
3. Color the first k vertices in the ordering red, and the remaining ones blue:
   - A shortest path from 1 to n now only switches between red and blue once, so every shortest path on 3 or more vertices must have a monochromatic edge.
4. Special case: this does not work if there is a direct edge from 1 to n.
   - Since graph is vertex-weighted, edge from 1 to n is the *only* shortest path.
   - We only need to make sure 1 and n get the same color.
   - Always possible, except if $n = 2$ and $k = 1$.

C — Customs Controls

Problem

Given a vertex-weighted graph, color k vertices red and $n - k$ vertices blue such that every shortest path from 1 to n has a monochromatic edge.

Solution

1. Dijkstra's algorithm finds all edges that are part of some shortest path from 1 to n.
2. These edges form a directed acyclic graph. Find a topological ordering.
3. Color the first k vertices in the ordering red, and the remaining ones blue:
   - A shortest path from 1 to n now only switches between red and blue once, so every shortest path on 3 or more vertices must have a monochromatic edge.
4. Special case: this does not work if there is a direct edge from 1 to n.
   - Since graph is vertex-weighted, edge from 1 to n is the *only* shortest path.
   - We only need to make sure 1 and n get the same color.
   - Always possible, except if $n = 2$ and $k = 1$.

Statistics at 4-hour mark: 67 submissions, 15 accepted, first after 01:39

H — Hiring Help

Problem

Given productivity values ℓ_i, f_i of n coders and productivity ℓ, f of consultant for t-hour long project, is there a weighted average of coders such that $\ell_{\text{avg}} \geq \ell/t$ and $f_{\text{avg}} \geq f/t$? Handle many queries like this, interleaved with some of the n coders leaving.

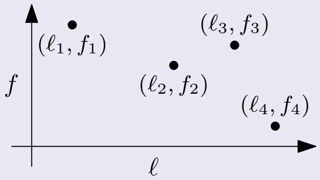
H — Hiring Help

Problem

Given productivity values ℓ_i, f_i of n coders and productivity ℓ, f of consultant for t-hour long project, is there a weighted average of coders such that $\ell_{\text{avg}} \geq \ell/t$ and $f_{\text{avg}} \geq f/t$? Handle many queries like this, interleaved with some of the n coders leaving.

Geometric View

1. The coders are a set of points P in 2D.



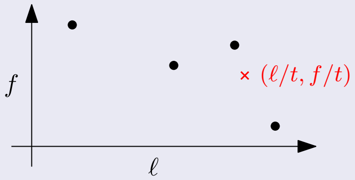
H — Hiring Help

Problem

Given productivity values ℓ_i, f_i of n coders and productivity ℓ, f of consultant for t-hour long project, is there a weighted average of coders such that $\ell_{\text{avg}} \geq \ell/t$ and $f_{\text{avg}} \geq f/t$? Handle many queries like this, interleaved with some of the n coders leaving.

Geometric View

1. The coders are a set of points P in 2D.
2. The consultant query is another point q in 2D.



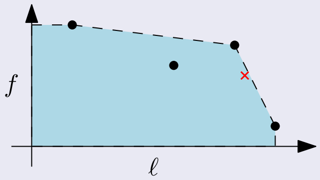
H — Hiring Help

Problem

Given productivity values ℓ_i, f_i of n coders and productivity ℓ, f of consultant for t-hour long project, is there a weighted average of coders such that $\ell_{\text{avg}} \geq \ell / t$ and $f_{\text{avg}} \geq f / t$? Handle many queries like this, interleaved with some of the n coders leaving.

Geometric View

1. The coders are a set of points P in 2D.
2. The consultant query is another point q in 2D.
3. The weighted average exists if and only if q is below the upper side of the convex hull of P.



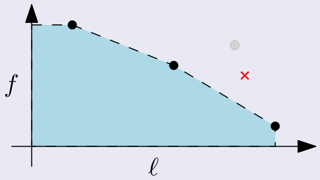
H — Hiring Help

Problem

Given productivity values ℓ_i, f_i of n coders and productivity ℓ, f of consultant for t-hour long project, is there a weighted average of coders such that $\ell_{avg} \geq \ell/t$ and $f_{avg} \geq f/t$? Handle many queries like this, interleaved with some of the n coders leaving.

Geometric View

1. The coders are a set of points P in 2D.
2. The consultant query is another point q in 2D.
3. The weighted average exists if and only if q is below the upper side of the convex hull of P.
4. Coders leaving corresponds to points being removed, leading to the convex hull changing.



H — Hiring Help

Reformulated Problem

Given set of points (x, y), maintain upper side of its convex hull, under removals of points and queries about whether other points (x^*, y^*) are below the hull.

H — Hiring Help

Reformulated Problem

Given set of points (x, y), maintain upper side of its convex hull, under removals of points and queries about whether other points (x^*, y^*) are below the hull.

Solution

1. The hull is a piecewise linear function, represent it as a sorted set of points $(x_1, y_1), (x_2, y_2), \ldots, (x_t, y_t)$ where $x_1 < x_2 < \ldots x_t$ and $y_1 > y_2 > \ldots y_t$.

H — Hiring Help

Reformulated Problem

Given set of points (x, y), maintain upper side of its convex hull, under removals of points and queries about whether other points (x^*, y^*) are below the hull.

Solution

1. The hull is a piecewise linear function, represent it as a sorted set of points $(x_1, y_1), (x_2, y_2), \ldots, (x_t, y_t)$ where $x_1 < x_2 < \ldots x_t$ and $y_1 > y_2 > \ldots y_t$.
2. For a query (x^*, y^*), find index i such that $x_{i-1} < x^* \leq x_i$ and check if (x^*, y^*) is below line from (x_{i-1}, y_{i-1}) to (x_i, y_i).

H — Hiring Help

Reformulated Problem

Given set of points (x, y), maintain upper side of its convex hull, under removals of points and queries about whether other points (x^*, y^*) are below the hull.

Solution

1. The hull is a piecewise linear function, represent it as a sorted set of points $(x_1, y_1), (x_2, y_2), \ldots, (x_t, y_t)$ where $x_1 < x_2 < \ldots x_t$ and $y_1 > y_2 > \ldots y_t$.
2. For a query (x^*, y^*), find index i such that $x_{i-1} < x^* \leq x_i$ and check if (x^*, y^*) is below line from (x_{i-1}, y_{i-1}) to (x_i, y_i).
3. Handling removals can be done,

H — Hiring Help

Reformulated Problem

Given set of points (x, y), maintain upper side of its convex hull, under *additions* of points and queries about whether other points (x^*, y^*) are below the hull.

Solution

1. The hull is a piecewise linear function, represent it as a sorted set of points $(x_1, y_1), (x_2, y_2), \ldots, (x_t, y_t)$ where $x_1 < x_2 < \ldots x_t$ and $y_1 > y_2 > \ldots y_t$.
2. For a query (x^*, y^*), find index i such that $x_{i-1} < x^* \leq x_i$ and check if (x^*, y^*) is below line from (x_{i-1}, y_{i-1}) to (x_i, y_i).
3. Handling removals can be done, but if we instead run the events in reverse order, the removals become *additions*, which are easier to handle.

H — Hiring Help

Reformulated Problem

Given set of points (x, y), maintain upper side of its convex hull, under *additions* of points and queries about whether other points (x^*, y^*) are below the hull.

Handling additions

1. If point to add is outside current hull, add it to our current set of points.

H — Hiring Help

Reformulated Problem

Given set of points (x, y), maintain upper side of its convex hull, under *additions* of points and queries about whether other points (x^*, y^*) are below the hull.

Handling additions

1. If point to add is outside current hull, add it to our current set of points.
2. Remove any concavities formed to left and right of new point.

H — Hiring Help

Reformulated Problem

Given set of points (x, y), maintain upper side of its convex hull, under *additions* of points and queries about whether other points (x^*, y^*) are below the hull.

Handling additions

1. If point to add is outside current hull, add it to our current set of points.
2. Remove any concavities formed to left and right of new point.
3. Issue(?): addition may take a long time because many old points could be discarded from hull.

H — Hiring Help

Reformulated Problem

Given set of points (x, y), maintain upper side of its convex hull, under *additions* of points and queries about whether other points (x^*, y^*) are below the hull.

Handling additions

1. If point to add is outside current hull, add it to our current set of points.
2. Remove any concavities formed to left and right of new point.
3. Issue(?): addition may take a long time because many old points could be discarded from hull.
4. Not an issue: once a point is discarded, it can never be added back, so total number of removals for all events is $\leq n$.

H — Hiring Help

Reformulated Problem

Given set of points (x, y), maintain upper side of its convex hull, under *additions* of points and queries about whether other points (x^*, y^*) are below the hull.

Handling additions

1. If point to add is outside current hull, add it to our current set of points.
2. Remove any concavities formed to left and right of new point.
3. Issue(?): addition may take a long time because many old points could be discarded from hull.
4. Not an issue: once a point is discarded, it can never be added back, so total number of removals for all events is $\leq n$.
5. $O((n+e)\log n)$ total time complexity.

H — Hiring Help

Reformulated Problem

Given set of points (x, y), maintain upper side of its convex hull, under *additions* of points and queries about whether other points (x^*, y^*) are below the hull.

Handling additions

1. If point to add is outside current hull, add it to our current set of points.
2. Remove any concavities formed to left and right of new point.
3. Issue(?): addition may take a long time because many old points could be discarded from hull.
4. Not an issue: once a point is discarded, it can never be added back, so total number of removals for all events is $\leq n$.
5. $O((n + e) \log n)$ total time complexity.

Statistics at 4-hour mark: 24 submissions, 7 accepted, first after 01:42

I — Intact Intervals

Problem

Given circular array a, how many ways can it be separated into two or more intervals such that array b can be obtained by permuting each interval separately?

Problem

Given circular array a, how many ways can it be separated into two or more intervals such that array b can be obtained by permuting each interval separately?

Solution for the **non-circular** case

1. Let $A_i = a_1, \ldots, a_i$ be the prefix of first i values of a.

Problem

Given circular array a, how many ways can it be separated into two or more intervals such that array b can be obtained by permuting each interval separately?

Solution for the **non-circular** case

1. Let $A_i = a_1, \ldots, a_i$ be the prefix of first i values of a.
2. Let s be the number of indices $1 \leq i < n$ such that A_i is a permutation of B_i.

Problem

Given circular array a, how many ways can it be separated into two or more intervals such that array b can be obtained by permuting each interval separately?

Solution for the **non-circular** case

1. Let $A_i = a_1, \ldots, a_i$ be the prefix of first i values of a.
2. Let s be the number of indices $1 \leq i < n$ such that A_i is a permutation of B_i.
3. Then the number of ways is $2^s - 1$:

I — Intact Intervals

Problem

Given circular array a, how many ways can it be separated into two or more intervals such that array b can be obtained by permuting each interval separately?

Solution for the **non-circular** case

1. Let $A_i = a_1, \ldots, a_i$ be the prefix of first i values of a.
2. Let s be the number of indices $1 \leq i < n$ such that A_i is a permutation of B_i.
3. Then the number of ways is $2^s - 1$:
   - breaking up a at any non-empty subset of these indices results in a valid separation

I — Intact Intervals

Problem

Given circular array a, how many ways can it be separated into two or more intervals such that array b can be obtained by permuting each interval separately?

Solution for the **non-circular** case

1. Let $A_i = a_1, \ldots, a_i$ be the prefix of first i values of a.
2. Let s be the number of indices $1 \leq i < n$ such that A_i is a permutation of B_i.
3. Then the number of ways is $2^s - 1$:
   - breaking up a at any non-empty subset of these indices results in a valid separation
   - breaking at any other index results in an invalid separation

I — Intact Intervals

Problem

Given circular array a, how many ways can it be separated into two or more intervals such that array b can be obtained by permuting each interval separately?

Solution for the **non-circular** case

1. Let $A_i = a_1, \ldots, a_i$ be the prefix of first i values of a.
2. Let s be the number of indices $1 \leq i < n$ such that A_i is a permutation of B_i.
3. Then the number of ways is $2^s - 1$:
   - breaking up a at any non-empty subset of these indices results in a valid separation
   - breaking at any other index results in an invalid separation
4. To find s quickly, can use a permutation-invariant hash function.

Problem

Given circular array a, how many ways can it be separated into two or more intervals such that array b can be obtained by permuting each interval separately?

Solution for the **non-circular** case

1. Let $A_i = a_1, \ldots, a_i$ be the prefix of first i values of a.
2. Let s be the number of indices $1 \leq i < n$ such that A_i is a permutation of B_i.
3. Then the number of ways is $2^s - 1$:
   - breaking up a at any non-empty subset of these indices results in a valid separation
   - breaking at any other index results in an invalid separation
4. To find s quickly, can use a permutation-invariant hash function.
   - Assign a random hash value $h(x)$ to each array value x.

Problem

Given circular array a, how many ways can it be separated into two or more intervals such that array b can be obtained by permuting each interval separately?

Solution for the **non-circular** case

1. Let $A_i = a_1, \ldots, a_i$ be the prefix of first i values of a.
2. Let s be the number of indices $1 \leq i < n$ such that A_i is a permutation of B_i.
3. Then the number of ways is $2^s - 1$:
   - breaking up a at any non-empty subset of these indices results in a valid separation
   - breaking at any other index results in an invalid separation
4. To find s quickly, can use a permutation-invariant hash function.
   - Assign a random hash value $h(x)$ to each array value x.
   - Define hash $h(A_i)$ of a prefix to be $\sum_{j=1}^{i} h(a_j)$.

I — Intact Intervals

Problem

Given circular array a, how many ways can it be separated into two or more intervals such that array b can be obtained by permuting each interval separately?

Solution for the **non-circular** case

1. Let $A_i = a_1, \ldots, a_i$ be the prefix of first i values of a.
2. Let s be the number of indices $1 \leq i < n$ such that A_i is a permutation of B_i.
3. Then the number of ways is $2^s - 1$:
   - breaking up a at any non-empty subset of these indices results in a valid separation
   - breaking at any other index results in an invalid separation
4. To find s quickly, can use a permutation-invariant hash function.
   - Assign a random hash value $h(x)$ to each array value x.
   - Define hash $h(A_i)$ of a prefix to be $\sum_{j=1}^{i} h(a_j)$.
   - If no hash collisions then A_i is a permutation of B_i if and only if $h(A_i) = h(B_i)$.

Problem

Given circular array a, how many ways can it be separated into two or more intervals such that array b can be obtained by permuting each interval separately?

Solution for the circular case

1. For each hash value z, let $s(z)$ be number of indices $0 \leq i < n$ such that $h(A_i) - h(B_i) = z$.

I — Intact Intervals

Problem

Given circular array a, how many ways can it be separated into two or more intervals such that array b can be obtained by permuting each interval separately?

Solution for the circular case

1. For each hash value z, let $s(z)$ be number of indices $0 \leq i < n$ such that $h(A_i) - h(B_i) = z$.
2. Then (assuming no hash collisions), the number of ways is $\sum_z 2^{s(z)} - s(z) - 1$.

Problem

Given circular array a, how many ways can it be separated into two or more intervals such that array b can be obtained by permuting each interval separately?

Solution for the circular case

1. For each hash value z, let $s(z)$ be number of indices $0 \leq i < n$ such that $h(A_i) - h(B_i) = z$.
2. Then (assuming no hash collisions), the number of ways is $\sum_z 2^{s(z)} - s(z) - 1$.
   - For each z, taking any subset of at least 2 of the $s(z)$ indices is a valid separation.

Problem

Given circular array a, how many ways can it be separated into two or more intervals such that array b can be obtained by permuting each interval separately?

Solution for the circular case

1. For each hash value z, let $s(z)$ be number of indices $0 \leq i < n$ such that $h(A_i) - h(B_i) = z$.
2. Then (assuming no hash collisions), the number of ways is $\sum_z 2^{s(z)} - s(z) - 1$.
   - For each z, taking any subset of at least 2 of the $s(z)$ indices is a valid separation.
   - Taking two indices with different values of $h(A_i) - h(B_i)$ gives an invalid separation.

Problem

Given circular array a, how many ways can it be separated into two or more intervals such that array b can be obtained by permuting each interval separately?

Solution for the circular case

1. For each hash value z, let $s(z)$ be number of indices $0 \leq i < n$ such that $h(A_i) - h(B_i) = z$.
2. Then (assuming no hash collisions), the number of ways is $\sum_z 2^{s(z)} - s(z) - 1$.
   - For each z, taking any subset of at least 2 of the $s(z)$ indices is a valid separation.
   - Taking two indices with different values of $h(A_i) - h(B_i)$ gives an invalid separation.
3. Results in an $O(n)$ time solution, assuming $O(1)$-time dictionaries which can be used to store the frequency of each hash value.

I — Intact Intervals

Problem

Given circular array a, how many ways can it be separated into two or more intervals such that array b can be obtained by permuting each interval separately?

Solution for the circular case

1. For each hash value z, let $s(z)$ be number of indices $0 \leq i < n$ such that $h(A_i) - h(B_i) = z$.
2. Then (assuming no hash collisions), the number of ways is $\sum_z 2^{s(z)} - s(z) - 1$.
   - For each z, taking any subset of at least 2 of the $s(z)$ indices is a valid separation.
   - Taking two indices with different values of $h(A_i) - h(B_i)$ gives an invalid separation.
3. Results in an $O(n)$ time solution, assuming $O(1)$-time dictionaries which can be used to store the frequency of each hash value.

Statistics at 4-hour mark: 11 submissions, 5 accepted, first after 01:51

B — Breaking Bars

Problem

Given list of rectangular chocolate bars, make as few breaks as possible to produce two equal collections of bars, each collection having at least t squares.

B — Breaking Bars

Problem

Given list of rectangular chocolate bars, make as few breaks as possible to produce two equal collections of bars, each collection having at least t squares.

Incorrect solution **for partitioning all the chocolate**

1. Compute the number of bars of each bar size (there are 21 sizes).

B — Breaking Bars

Problem

Given list of rectangular chocolate bars, make as few breaks as possible to produce two equal collections of bars, each collection having at least t squares.

Incorrect solution **for partitioning all the chocolate**

1. Compute the number of bars of each bar size (there are 21 sizes).
2. Always greedily pair up bars so that you never have 2 or more of any size.

B — Breaking Bars

Problem

Given list of rectangular chocolate bars, make as few breaks as possible to produce two equal collections of bars, each collection having at least t squares.

Incorrect solution **for partitioning all the chocolate**

1. Compute the number of bars of each bar size (there are 21 sizes).
2. Always greedily pair up bars so that you never have 2 or more of any size.
3. Find minimum number of breaks needed for remaining bars.

B — Breaking Bars

Problem

Given list of rectangular chocolate bars, make as few breaks as possible to produce two equal collections of bars, each collection having at least t squares.

Incorrect solution **for partitioning all the chocolate**

1. Compute the number of bars of each bar size (there are 21 sizes).
2. Always greedily pair up bars so that you never have 2 or more of any size.
3. Find minimum number of breaks needed for remaining bars.
4. Can be computed efficiently by dynamic programming over 2^{21} states.

B — Breaking Bars

Problem

Given list of rectangular chocolate bars, make as few breaks as possible to produce two equal collections of bars, each collection having at least t squares.

Incorrect solution **for partitioning all the chocolate**

1. Compute the number of bars of each bar size (there are 21 sizes).
2. Always greedily pair up bars so that you never have 2 or more of any size.
3. Find minimum number of breaks needed for remaining bars.
4. Can be computed efficiently by dynamic programming over 2^{21} states.
5. Does not always give the optimum number of breaks. Example:

   `3x2 3x3 1x5 2x5 3x5 3x5`

   Split one 3x5 as `3x2+3x3` and the other as `1x5+2x5` to get away with two splits.

B — Breaking Bars

Problem

Given list of rectangular chocolate bars, make as few breaks as possible to produce two equal collections of bars, each collection having at least t squares.

Incorrect solution **for partitioning all the chocolate**

1. Compute the number of bars of each bar size (there are 21 sizes).
2. Always greedily pair up bars so that you never have 2 or more of any size.
3. Find minimum number of breaks needed for remaining bars.
4. Can be computed efficiently by dynamic programming over 2^{21} states.
5. Does not always give the optimum number of breaks. Example:

   3x2 3x3 1x5 2x5 3x5 3x5

   Split one 3x5 as 3x2+3x3 and the other as 1x5+2x5 to get away with two splits.
6. But this *does* give upper bound on number of breaks that may be needed (it is 9).

B — Breaking Bars

Problem

Given list of rectangular chocolate bars, make as few breaks as possible to produce two equal collections of bars, each collection having at least t squares.

Actual Solution

1. Recursively search for the best way to break the bars, going from larger bars to smaller ones.

B — Breaking Bars

Problem

Given list of rectangular chocolate bars, make as few breaks as possible to produce two equal collections of bars, each collection having at least t squares.

Actual Solution

1. Recursively search for the best way to break the bars, going from larger bars to smaller ones.
2. Keep track of number of breaks made and how many squares can be partitioned among the larger bars already considered.

B — Breaking Bars

Problem

Given list of rectangular chocolate bars, make as few breaks as possible to produce two equal collections of bars, each collection having at least t squares.

Actual Solution

1. Recursively search for the best way to break the bars, going from larger bars to smaller ones.
2. Keep track of number of breaks made and how many squares can be partitioned among the larger bars already considered.
3. Avoid recursive calls that will produce more breaks than best solution found.

B — Breaking Bars

Problem

Given list of rectangular chocolate bars, make as few breaks as possible to produce two equal collections of bars, each collection having at least t squares.

Actual Solution

1. Recursively search for the best way to break the bars, going from larger bars to smaller ones.
2. Keep track of number of breaks made and how many squares can be partitioned among the larger bars already considered.
3. Avoid recursive calls that will produce more breaks than best solution found.
4. Since there is no need for more than 9 breaks and any bar be broken in at most 5 different ways, this turns out to be fast enough.

B — Breaking Bars

Problem

Given list of rectangular chocolate bars, make as few breaks as possible to produce two equal collections of bars, each collection having at least t squares.

Actual Solution

1. Recursively search for the best way to break the bars, going from larger bars to smaller ones.
2. Keep track of number of breaks made and how many squares can be partitioned among the larger bars already considered.
3. Avoid recursive calls that will produce more breaks than best solution found.
4. Since there is no need for more than 9 breaks and any bar be broken in at most 5 different ways, this turns out to be fast enough.

Statistics at 4-hour mark: 17 submissions, 1 accepted, first after 01:13

E — Eavesdropper Evasion

Problem

Find a way to schedule sending n messages as fast as possible but so that at most 2 of them are fully sent within any time frame of length x.

E — Eavesdropper Evasion

Problem

Find a way to schedule sending n messages as fast as possible but so that at most 2 of them are fully sent within any time frame of length x.

Solution

1. Messages longer than x can be ignored as they can never be intercepted

E — Eavesdropper Evasion

Problem

Find a way to schedule sending n messages as fast as possible but so that at most 2 of them are fully sent within any time frame of length x.

Solution

1. Messages longer than x can be ignored as they can never be intercepted
   - Mostly - final result can not be smaller than the longest message!

E — Eavesdropper Evasion

Problem

Find a way to schedule sending n messages as fast as possible but so that at most 2 of them are fully sent within any time frame of length x.

Solution

1. Messages longer than x can be ignored as they can never be intercepted
   - Mostly - final result can not be smaller than the longest message!
2. Solve the case if at most 1 message was allowed to be intercepted

E — Eavesdropper Evasion

Problem

Find a way to schedule sending n messages as fast as possible but so that at most 2 of them are fully sent within any time frame of length x.

Solution

1. Messages longer than x can be ignored as they can never be intercepted
   - Mostly - final result can not be smaller than the longest message!
2. Solve the case if at most 1 message was allowed to be intercepted
3. Generalize it to at most 2 intercepted messages

E — Eavesdropper Evasion

Problem

Find a way to schedule sending n messages as fast as possible but so that at most 2 of them are fully sent within any time frame of length x.

Solution if at most 1 message is allowed to be intercepted

1. If a message (regardless of length) starts at time a, the *next* message can start no earlier than time $a + x + 1 - t$, where t is the length of the *next* message.

Problem

Find a way to schedule sending n messages as fast as possible but so that at most 2 of them are fully sent within any time frame of length x.

Solution if at most 1 message is allowed to be intercepted

1. If a message (regardless of length) starts at time a, the *next* message can start no earlier than time $a + x + 1 - t$, where t is the length of the *next* message.
2. Repeating this, the total time to send all n messages becomes

$$x + 1 + \sum_{i=2}^{n-1}(x + 1 - t_i),$$

where t_i is the length of the ith message sent.

E — Eavesdropper Evasion

Problem

Find a way to schedule sending n messages as fast as possible but so that at most 2 of them are fully sent within any time frame of length x.

Solution if at most 1 message is allowed to be intercepted

1. If a message (regardless of length) starts at time a, the *next* message can start no earlier than time $a + x + 1 - t$, where t is the length of the *next* message.
2. Repeating this, the total time to send all n messages becomes
$$x + 1 + \sum_{i=2}^{n-1} (x + 1 - t_i),$$
where t_i is the length of the ith message sent.
3. Note: length of the first and last message does not affect the total send time!

E — Eavesdropper Evasion

Problem

Find a way to schedule sending n messages as fast as possible but so that at most 2 of them are fully sent within any time frame of length x.

Solution if at most 1 message is allowed to be intercepted

1. If a message (regardless of length) starts at time a, the *next* message can start no earlier than time $a + x + 1 - t$, where t is the length of the *next* message.
2. Repeating this, the total time to send all n messages becomes

$$x + 1 + \sum_{i=2}^{n-1} (x + 1 - t_i),$$

   where t_i is the length of the ith message sent.
3. Note: length of the first and last message does not affect the total send time!
4. By placing the two shortest messages first and last, we get the optimal solution.

Solution for the 2-message case

1. Key observation: we can view this as having 2 separate channels, and we want at most 1 intercepted message in each channel (not immediately obvious!).

Solution for the 2-message case

1. Key observation: we can view this as having 2 separate channels, and we want at most 1 intercepted message in each channel (not immediately obvious!).
2. Only thing to decide is which messages to send on which channel.

E — Eavesdropper Evasion

Solution for the 2-message case

1. Key observation: we can view this as having 2 separate channels, and we want at most 1 intercepted message in each channel (not immediately obvious!).
2. Only thing to decide is which messages to send on which channel.
   - Channel with r msgs of lengths $t_1, \ldots, t_r$ finishes in time $x + 1 + \sum_{i=2}^{r-1}(x + 1 - t_i)$.

E — Eavesdropper Evasion

Solution for the 2-message case

1. Key observation: we can view this as having 2 separate channels, and we want at most 1 intercepted message in each channel (not immediately obvious!).
2. Only thing to decide is which messages to send on which channel.
   - Channel with r msgs of lengths $t_1, \ldots, t_r$ finishes in time $x + 1 + \sum_{i=2}^{r-1}(x + 1 - t_i)$.
   - Take the 4 shortest messages and put first and last in each channel.

E — Eavesdropper Evasion

Solution for the 2-message case

1. Key observation: we can view this as having 2 separate channels, and we want at most 1 intercepted message in each channel (not immediately obvious!).
2. Only thing to decide is which messages to send on which channel.
   - Channel with r msgs of lengths $t_1, \ldots, t_r$ finishes in time $x + 1 + \sum_{i=2}^{r-1}(x + 1 - t_i)$.
   - Take the 4 shortest messages and put first and last in each channel.
   - Each remaining message of length t gets a weight $w = x + 1 - t$.

E — Eavesdropper Evasion

Solution for the 2-message case

1. Key observation: we can view this as having 2 separate channels, and we want at most 1 intercepted message in each channel (not immediately obvious!).
2. Only thing to decide is which messages to send on which channel.
   - Channel with r msgs of lengths $t_1, \ldots, t_r$ finishes in time $x + 1 + \sum_{i=2}^{r-1} (x + 1 - t_i)$.
   - Take the 4 shortest messages and put first and last in each channel.
   - Each remaining message of length t gets a weight $w = x + 1 - t$.
3. We have $m = n - 4$ weights $w_1, \ldots, w_m$ of total weight $W = w_1 + \ldots + w_n$.

Solution for the 2-message case

1. Key observation: we can view this as having 2 separate channels, and we want at most 1 intercepted message in each channel (not immediately obvious!).
2. Only thing to decide is which messages to send on which channel.
   - Channel with r msgs of lengths $t_1, \ldots, t_r$ finishes in time $x + 1 + \sum_{i=2}^{r-1}(x + 1 - t_i)$.
   - Take the 4 shortest messages and put first and last in each channel.
   - Each remaining message of length t gets a weight $w = x + 1 - t$.
3. We have $m = n - 4$ weights $w_1, \ldots, w_m$ of total weight $W = w_1 + \ldots + w_n$.
4. Want to find a subset $S \subseteq [m]$ such that $\sum_{i \in S} w_i$ is as close to $W/2$ as possible.

E — Eavesdropper Evasion

Solution for the 2-message case

1. Key observation: we can view this as having 2 separate channels, and we want at most 1 intercepted message in each channel (not immediately obvious!).
2. Only thing to decide is which messages to send on which channel.
   - Channel with r msgs of lengths $t_1, \ldots, t_r$ finishes in time $x + 1 + \sum_{i=2}^{r-1}(x + 1 - t_i)$.
   - Take the 4 shortest messages and put first and last in each channel.
   - Each remaining message of length t gets a weight $w = x + 1 - t$.
3. We have $m = n - 4$ weights $w_1, \ldots, w_m$ of total weight $W = w_1 + \ldots + w_n$.
4. Want to find a subset $S \subseteq [m]$ such that $\sum_{i \in S} w_i$ is as close to $W/2$ as possible.
5. This is a Subset Sum/Knapsack problem.

E — Eavesdropper Evasion

Solution for the 2-message case

1. Key observation: we can view this as having 2 separate channels, and we want at most 1 intercepted message in each channel (not immediately obvious!).
2. Only thing to decide is which messages to send on which channel.
   - Channel with r msgs of lengths $t_1, \ldots, t_r$ finishes in time $x + 1 + \sum_{i=2}^{r-1}(x + 1 - t_i)$.
   - Take the 4 shortest messages and put first and last in each channel.
   - Each remaining message of length t gets a weight $w = x + 1 - t$.
3. We have $m = n - 4$ weights $w_1, \ldots, w_m$ of total weight $W = w_1 + \ldots + w_n$.
4. Want to find a subset $S \subseteq [m]$ such that $\sum_{i \in S} w_i$ is as close to $W/2$ as possible.
5. This is a Subset Sum/Knapsack problem.
   - Classic dynprog algorithm solves it in time $O(nW) = O(n^2 x)$. Too slow.

E — Eavesdropper Evasion

Solution for the 2-message case

1. Key observation: we can view this as having 2 separate channels, and we want at most 1 intercepted message in each channel (not immediately obvious!).
2. Only thing to decide is which messages to send on which channel.
   - Channel with r msgs of lengths $t_1, \ldots, t_r$ finishes in time $x + 1 + \sum_{i=2}^{r-1}(x + 1 - t_i)$.
   - Take the 4 shortest messages and put first and last in each channel.
   - Each remaining message of length t gets a weight $w = x + 1 - t$.
3. We have $m = n - 4$ weights $w_1, \ldots, w_m$ of total weight $W = w_1 + \ldots + w_n$.
4. Want to find a subset $S \subseteq [m]$ such that $\sum_{i \in S} w_i$ is as close to $W/2$ as possible.
5. This is a Subset Sum/Knapsack problem.
   - Classic dynprog algorithm solves it in time $O(nW) = O(n^2 x)$. Too slow.
   - Can be solved in $O(n \cdot x)$ time (Pisinger, 1999).

E — Eavesdropper Evasion

Solution for the 2-message case

1. Key observation: we can view this as having 2 separate channels, and we want at most 1 intercepted message in each channel (not immediately obvious!).
2. Only thing to decide is which messages to send on which channel.
   - Channel with r msgs of lengths $t_1, \ldots, t_r$ finishes in time $x + 1 + \sum_{i=2}^{r-1}(x + 1 - t_i)$.
   - Take the 4 shortest messages and put first and last in each channel.
   - Each remaining message of length t gets a weight $w = x + 1 - t$.
3. We have $m = n - 4$ weights $w_1, \ldots, w_m$ of total weight $W = w_1 + \ldots + w_n$.
4. Want to find a subset $S \subseteq [m]$ such that $\sum_{i \in S} w_i$ is as close to $W/2$ as possible.
5. This is a Subset Sum/Knapsack problem.
   - Classic dynprog algorithm solves it in time $O(nW) = O(n^2 x)$. Too slow.
   - Can be solved in $O(n \cdot x)$ time (Pisinger, 1999).

Statistics at 4-hour mark: 19 submissions, 0 accepted

M — Marvelous Marathon

Problem

Find a path of length x subject to certain constraints in a $2 \times m$ grid so that the sum of the values in the cells of the path is maximized.

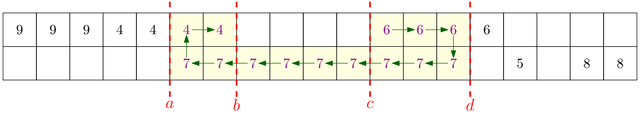
M — Marvelous Marathon

Problem

Find a path of length x subject to certain constraints in a $2 \times m$ grid so that the sum of the values in the cells of the path is maximized.

Formalized version of problem

Find integers $0 \leq a \leq b \leq c \leq d \leq m$ such that $2(b-a)+(c-b)+2(d-c)=x$.



M — Marvelous Marathon

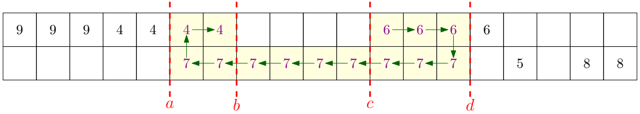
Problem

Find a path of length x subject to certain constraints in a $2 \times m$ grid so that the sum of the values in the cells of the path is maximized.

Formalized version of problem

Find integers $0 \leq a \leq b \leq c \leq d \leq m$ such that $2(b-a)+(c-b)+2(d-c)=x$.

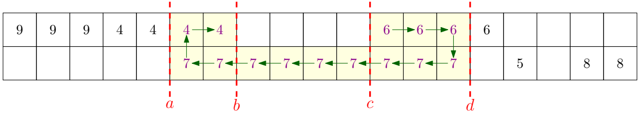
$$\text{Maximize} \left\{ \begin{array}{c} \text{sum of cells of one row in range } [a, d) \\ \text{plus} \\ \text{sum of cells of other row in ranges } [a, b) \text{ and } [c, d) \end{array} \right\}$$



M — Marvelous Marathon

Solution outline

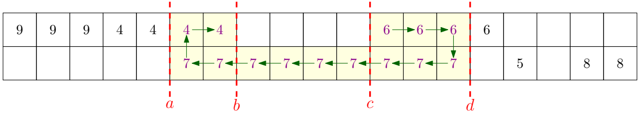
1. m is very large, cannot loop over all cells



M — Marvelous Marathon

Solution outline

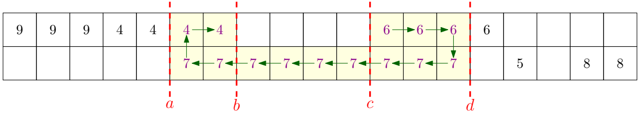
1. m is very large, cannot loop over all cells
2. Large parts of grid look the same because only n segments



M — Marvelous Marathon

Solution outline

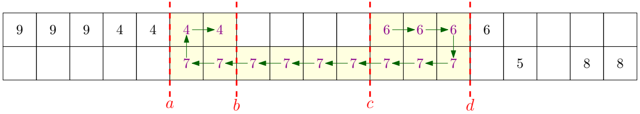
1. m is very large, cannot loop over all cells
2. Large parts of grid look the same because only n segments
3. Separately handle three main cases: 0, 1 or 2 U-turns



M — Marvelous Marathon

Solution outline

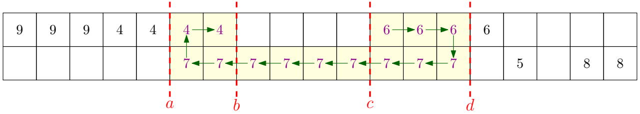
1. m is very large, cannot loop over all cells
2. Large parts of grid look the same because only n segments
3. Separately handle three main cases: 0, 1 or 2 U-turns
4. We focus here only on the hardest case with 2 U-turns.



M — Marvelous Marathon

Insight 1

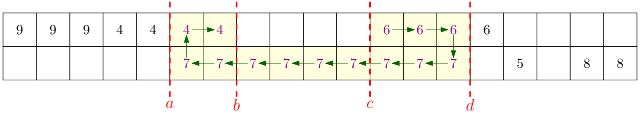
1. We can assume solution has the gap in the lower half
   - Run solution again on flipped input to cover opposite case



M — Marvelous Marathon

Insight 1

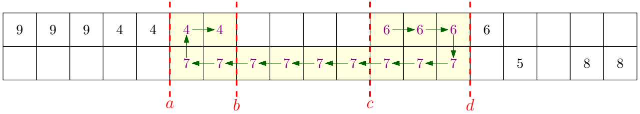
1. We can assume solution has the gap in the lower half
   - Run solution again on flipped input to cover opposite case
2. There is an optimal solution where a or d is at a segment endpoint (or 0 or m). Otherwise we could decrease (or increase) both a and d with 1 until either



M — Marvelous Marathon

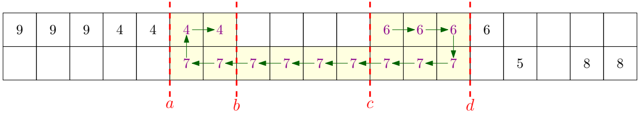
Insight 1

1. We can assume solution has the gap in the lower half
   - Run solution again on flipped input to cover opposite case
2. There is an optimal solution where a or d is at a segment endpoint (or 0 or m). Otherwise we could decrease (or increase) both a and d with 1 until either
   - a or d reaches a segment endpoint, *or*



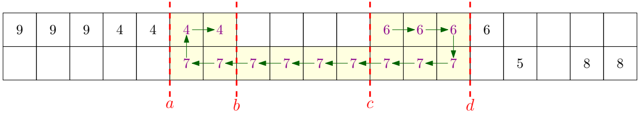
Insight 1

1. We can assume solution has the gap in the lower half
   - Run solution again on flipped input to cover opposite case
2. There is an optimal solution where a or d is at a segment endpoint (or 0 or m). Otherwise we could decrease (or increase) both a and d with 1 until either
   - a or d reaches a segment endpoint, *or*
   - $d = c$ or $a = b$, in which case we end up with the 1 U-turn case (handled separately, left as an exercise!)



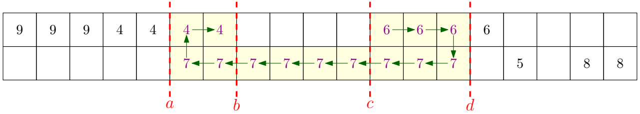
Insight 1

1. We can assume solution has the gap in the lower half
   - Run solution again on flipped input to cover opposite case
2. There is an optimal solution where a or d is at a segment endpoint (or 0 or m). Otherwise we could decrease (or increase) both a and d with 1 until either
   - a or d reaches a segment endpoint, *or*
   - $d = c$ or $a = b$, in which case we end up with the 1 U-turn case (handled separately, left as an exercise!)
3. We can assume a is the endpoint
   - Run solution again on reversed input to cover opposite case.



Insight 2

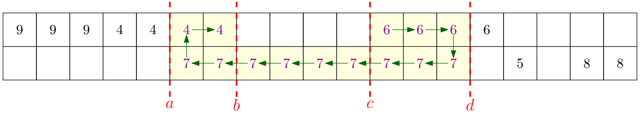
1. There is an optimal solution where b or c is at a segment endpoint, for the same reasoning as before (except that this time we would show it by shifting b or c).



M — Marvelous Marathon

Insight 2

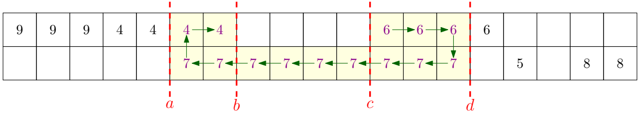
1. There is an optimal solution where b or c is at a segment endpoint, for the same reasoning as before (except that this time we would show it by shifting b or c).
2. We end up with two cases to consider:
   - a and b are segment endpoints
   - a and c are segment endpoints



M — Marvelous Marathon

Insight 2

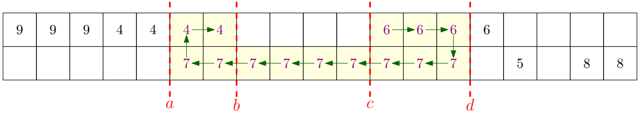
1. There is an optimal solution where b or c is at a segment endpoint, for the same reasoning as before (except that this time we would show it by shifting b or c).
2. We end up with two cases to consider:
   - a and b are segment endpoints
   - a and c are segment endpoints
3. Will focus on the first of these; the other must also be solved, but is done in a very similar fashion.



M — Marvelous Marathon

Sliding

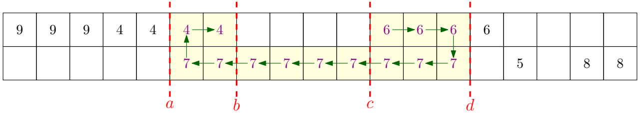
1. Fix some a and b. ($O(n^2)$ possible choices.)



M — Marvelous Marathon

Sliding

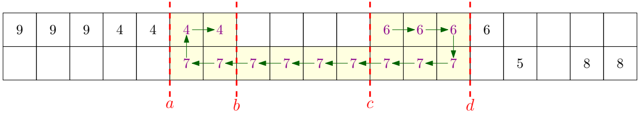
1. Fix some a and b. ($O(n^2)$ possible choices.)
2. Set $c = b$ (or $c = b + 1$ if x is odd) and $d = a + \lceil x/2 \rceil$



M — Marvelous Marathon

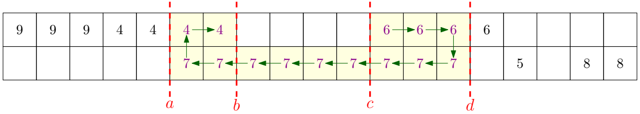
Sliding

1. Fix some a and b. ($O(n^2)$ possible choices.)
2. Set $c = b$ (or $c = b + 1$ if x is odd) and $d = a + \lceil x/2 \rceil$
3. Idea: slide c and d right (c twice as fast as d) and maintain current score.



Sliding

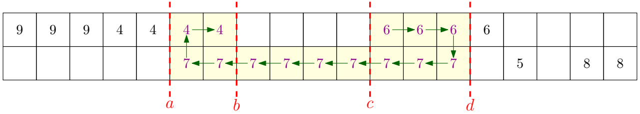
1. Fix some a and b. ($O(n^2)$ possible choices.)
2. Set $c = b$ (or $c = b + 1$ if x is odd) and $d = a + \lceil x/2 \rceil$
3. Idea: slide c and d right (c twice as fast as d) and maintain current score.
   - Since grid values rarely change value, we can slide many steps at a time.



M — Marvelous Marathon

Sliding

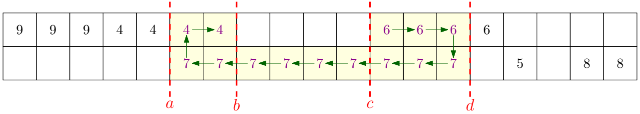
1. Fix some a and b. ($O(n^2)$ possible choices.)
2. Set $c = b$ (or $c = b + 1$ if x is odd) and $d = a + \lceil x/2 \rceil$
3. Idea: slide c and d right (c twice as fast as d) and maintain current score.
   - Since grid values rarely change value, we can slide many steps at a time.
   - Calculate when either c or d hits next segment endpoint and jump directly there.



M — Marvelous Marathon

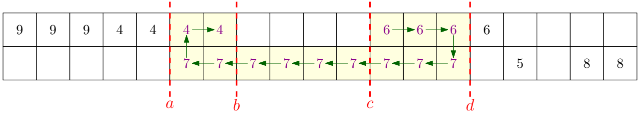
Sliding

1. Fix some a and b. ($O(n^2)$ possible choices.)
2. Set $c = b$ (or $c = b + 1$ if x is odd) and $d = a + \lceil x/2 \rceil$
3. Idea: slide c and d right (c twice as fast as d) and maintain current score.
   - Since grid values rarely change value, we can slide many steps at a time.
   - Calculate when either c or d hits next segment endpoint and jump directly there.
   - Repeat until c reached the end of the road.



Sliding

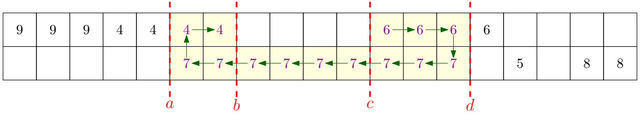
1. Fix some a and b. ($O(n^2)$ possible choices.)
2. Set $c = b$ (or $c = b + 1$ if x is odd) and $d = a + \lceil x/2 \rceil$
3. Idea: slide c and d right (c twice as fast as d) and maintain current score.
   - Since grid values rarely change value, we can slide many steps at a time.
   - Calculate when either c or d hits next segment endpoint and jump directly there.
   - Repeat until c reached the end of the road.
4. "Next segment endpoint" can be found in $O(1)$, for a total complexity of $O(n^3)$.



M — Marvelous Marathon

Sliding

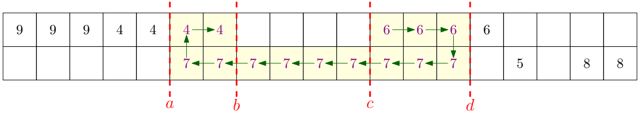
1. Fix some a and b. ($O(n^2)$ possible choices.)
2. Set $c = b$ (or $c = b + 1$ if x is odd) and $d = a + \lceil x/2 \rceil$
3. Idea: slide c and d right (c twice as fast as d) and maintain current score.
   - Since grid values rarely change value, we can slide many steps at a time.
   - Calculate when either c or d hits next segment endpoint and jump directly there.
   - Repeat until c reached the end of the road.
4. "Next segment endpoint" can be found in $O(1)$, for a total complexity of $O(n^3)$.
   - An optimized $O(n^4)$ implementation might also pass.



Sliding

1. Fix some a and b. ($O(n^2)$ possible choices.)
2. Set $c = b$ (or $c = b + 1$ if x is odd) and $d = a + \lceil x/2 \rceil$
3. Idea: slide c and d right (c twice as fast as d) and maintain current score.
   - Since grid values rarely change value, we can slide many steps at a time.
   - Calculate when either c or d hits next segment endpoint and jump directly there.
   - Repeat until c reached the end of the road.
4. "Next segment endpoint" can be found in $O(1)$, for a total complexity of $O(n^3)$.
   - An optimized $O(n^4)$ implementation might also pass.

Statistics at 4-hour mark: 0 submissions, 0 accepted



Results!