Delft Algorithm Programming Contest (DAPC) 2024

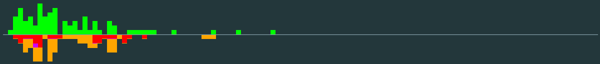
Solutions presentation

The BAPC 2024 jury

September 21, 2024

H: Human Pyramid

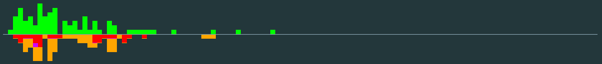
Problem author: Mees de Vries



Problem: Find the highest possible pyramid you can build with $n \leq 10^{12}$ people.

H: Human Pyramid

Problem author: Mees de Vries

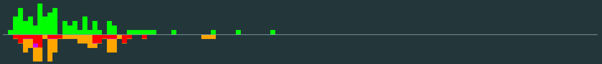


Problem: Find the highest possible pyramid you can build with $n \leq 10^{12}$ people.

Note: A human pyramid of height h consists of $p(h) = \dfrac{h \cdot (h+1)}{2}$ people.

H: Human Pyramid

Problem author: Mees de Vries



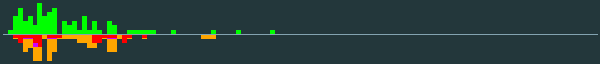
Problem: Find the highest possible pyramid you can build with $n \leq 10^{12}$ people.

Note: A human pyramid of height h consists of $p(h) = \dfrac{h \cdot (h+1)}{2}$ people.

Solution 1: Iterate over increasing values of h until you hit n. $\mathcal{O}(\sqrt{n})$.

H: Human Pyramid

Problem author: Mees de Vries



Problem: Find the highest possible pyramid you can build with $n \leq 10^{12}$ people.

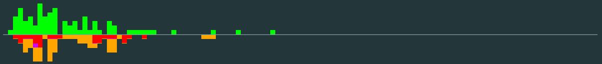
Note: A human pyramid of height h consists of $p(h) = \dfrac{h \cdot (h+1)}{2}$ people.

Solution 1: Iterate over increasing values of h until you hit n. $\mathcal{O}(\sqrt{n})$.

Solution 2: Binary search the height of the pyramid. $\mathcal{O}(\log n)$.

H: Human Pyramid

Problem author: Mees de Vries



Problem: Find the highest possible pyramid you can build with $n \leq 10^{12}$ people.

Note: A human pyramid of height h consists of $p(h) = \dfrac{h \cdot (h+1)}{2}$ people.

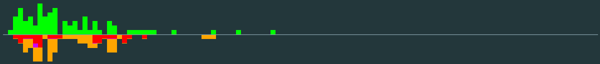
Solution 1: Iterate over increasing values of h until you hit n. $\mathcal{O}(\sqrt{n})$.

Solution 2: Binary search the height of the pyramid. $\mathcal{O}(\log n)$.

Solution 3: Invert the function p: $p^{-1}(n) = \left\lfloor \dfrac{\sqrt{8n+1}-1}{2} \right\rfloor$. $\mathcal{O}(1)$.

H: Human Pyramid

Problem author: Mees de Vries



Problem: Find the highest possible pyramid you can build with $n \leq 10^{12}$ people.

Note: A human pyramid of height h consists of $p(h) = \dfrac{h \cdot (h+1)}{2}$ people.

Solution 1: Iterate over increasing values of h until you hit n. $\mathcal{O}(\sqrt{n})$.

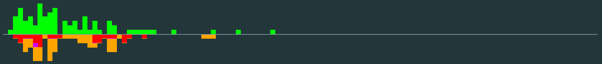
Solution 2: Binary search the height of the pyramid. $\mathcal{O}(\log n)$.

Solution 3: Invert the function p: $p^{-1}(n) = \left\lfloor \dfrac{\sqrt{8n+1} - 1}{2} \right\rfloor$. $\mathcal{O}(1)$.

This could give floating-point errors, but with these input limits and using `doubles`, it does not.

H: Human Pyramid

Problem author: Mees de Vries



Problem: Find the highest possible pyramid you can build with $n \leq 10^{12}$ people.

Note: A human pyramid of height h consists of $p(h) = \dfrac{h \cdot (h+1)}{2}$ people.

Solution 1: Iterate over increasing values of h until you hit n. $\mathcal{O}(\sqrt{n})$.

Solution 2: Binary search the height of the pyramid. $\mathcal{O}(\log n)$.

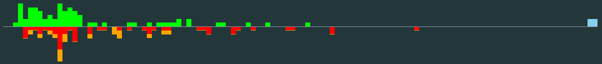
Solution 3: Invert the function p: $p^{-1}(n) = \left\lfloor \dfrac{\sqrt{8n+1}-1}{2} \right\rfloor$. $\mathcal{O}(1)$.

This could give floating-point errors, but with these input limits and using `doubles`, it does not.

Statistics: 143 submissions, 75 accepted

B: Battle of Nieuwpoort

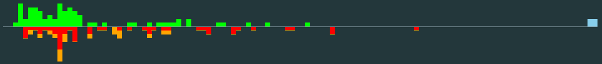
Problem author: Timon Knigge



Problem: Given a year y in decimal, with $2 \leq y \leq 2024$, if possible, find base b with $2 \leq b \leq 16$ such that when y is written in base-b, it ends with "00".

B: Battle of Nieuwpoort

Problem author: Timon Knigge

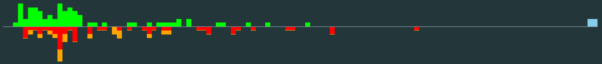


Problem: Given a year y in decimal, with $2 \leq y \leq 2024$, if possible, find base b with $2 \leq b \leq 16$ such that when y is written in base-b, it ends with "00".

Equivalently: Determine b such that b^2 divides y without remainder. So, just check for all $b \in \{2, \ldots, 16\}$ if $b^2 \mid y$. In fact, suffices to check the primes $b \in \{2, 3, 5, 7, 11, 13\}$.

B: Battle of Nieuwpoort

Problem author: Timon Knigge



Problem: Given a year y in decimal, with $2 \leq y \leq 2024$, if possible, find base b with $2 \leq b \leq 16$ such that when y is written in base-b, it ends with "00".

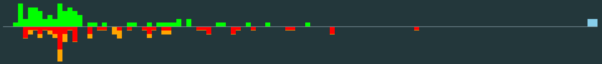
Equivalently: Determine b such that b^2 divides y without remainder. So, just check for all $b \in \{2, \ldots, 16\}$ if $b^2 \mid y$. In fact, suffices to check the primes $b \in \{2, 3, 5, 7, 11, 13\}$.

Solution (math): Check if $b^2 \mid y$ using integer modulus:

```
y % (b * b) == 0
```

B: Battle of Nieuwpoort

Problem author: Timon Knigge



Problem: Given a year y in decimal, with $2 \leq y \leq 2024$, if possible, find base b with $2 \leq b \leq 16$ such that when y is written in base-b, it ends with "00".

Equivalently: Determine b such that b^2 divides y without remainder. So, just check for all $b \in \{2, \ldots, 16\}$ if $b^2 \mid y$. In fact, suffices to check the primes $b \in \{2, 3, 5, 7, 11, 13\}$.

Solution (math): Check if $b^2 \mid y$ using integer modulus:

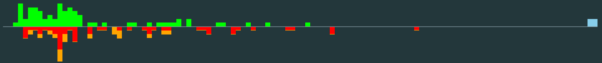
```
y % (b * b) == 0
```

Solution (string): Check if y written in base-b ends with "00". Some programming languages support this natively, such as Java's `Integer.toString(y, b)`. You can also do this digit by digit:

$\textit{letters} = $ "`0123456789abcdef`"
$s = $ ""
while $y > 0$:
    $s \mathrel{+}= \textit{letters}[y \;\%\; b]$
    $y = y/b$ (integer division)
return reversed(s)

B: Battle of Nieuwpoort

Problem author: Timon Knigge



Problem: Given a year y in decimal, with $2 \leq y \leq 2024$, if possible, find base b with $2 \leq b \leq 16$ such that when y is written in base-b, it ends with “00”.

Equivalently: Determine b such that b^2 divides y without remainder. So, just check for all $b \in \{2, \ldots, 16\}$ if $b^2 \mid y$. In fact, suffices to check the primes $b \in \{2, 3, 5, 7, 11, 13\}$.

Solution (math): Check if $b^2 \mid y$ using integer modulus:

```
y % (b * b) == 0
```

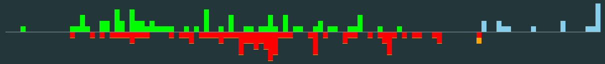
Solution (string): Check if y written in base-b ends with “00”. Some programming languages support this natively, such as Java's `Integer.toString(y, b)`. You can also do this digit by digit:

$letters =$ “`0123456789abcdef`”
$s =$ “”
while $y > 0$:
    $s \mathrel{+}= letters[y \;\%\; b]$
    $y = y/b$ (integer division)
return reversed(s)

Statistics: 142 submissions, 71 accepted, 4 unknown

E: Expected Error

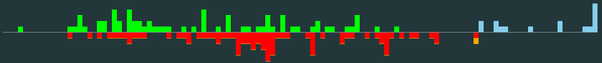
Problem author: Mike de Vries



Problem: You are typing your password, but your finger slipped and you are not sure whether you pressed a wrong key. Determine whether to continue typing, press backspace and continue typing or restart typing from scratch.

E: Expected Error

Problem author: Mike de Vries



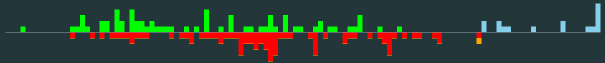
Problem: You are typing your password, but your finger slipped and you are not sure whether you pressed a wrong key. Determine whether to continue typing, press backspace and continue typing or restart typing from scratch.

Solution: Let us measure time in deciseconds to avoid decimals.

- If the password is wrong, this adds $4 + n$ deciseconds to your total time.
- We find `continue` yields an expected time of $1 + n - k + (4 + n)p/100$ deciseconds.
- We find `backspace` yields an expected time of $2 + n - k + (4 + n)(1 - p/100)$ deciseconds.
- We find `restart` yields an expected time of $4 + n$ deciseconds.
- To avoid decimals again, compare $100(1 + n - k) + (4 + n)p$ with $100(2 + n - k) + (4 + n)(100 - p)$ and $100(4 + n)$.
- The guarantee of a unique optimal strategy means one of these integers will be the smallest.

E: Expected Error

Problem author: Mike de Vries



Problem: You are typing your password, but your finger slipped and you are not sure whether you pressed a wrong key. Determine whether to continue typing, press backspace and continue typing or restart typing from scratch.

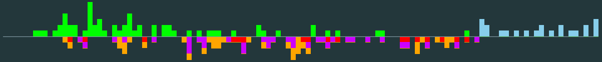
Solution: Let us measure time in deciseconds to avoid decimals.

- If the password is wrong, this adds $4 + n$ deciseconds to your total time.
- We find `continue` yields an expected time of $1 + n - k + (4 + n)p/100$ deciseconds.
- We find `backspace` yields an expected time of $2 + n - k + (4 + n)(1 - p/100)$ deciseconds.
- We find `restart` yields an expected time of $4 + n$ deciseconds.
- To avoid decimals again, compare $100(1 + n - k) + (4 + n)p$ with $100(2 + n - k) + (4 + n)(100 - p)$ and $100(4 + n)$.
- The guarantee of a unique optimal strategy means one of these integers will be the smallest.

Statistics: 165 submissions, 67 accepted, 16 unknown

D: Dialling Digits

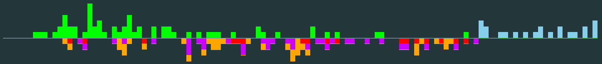
Problem author: Ragnar Groot Koerkamp



Problem: For each phone number, output the matching words.

D: Dialling Digits

Problem author: Ragnar Groot Koerkamp

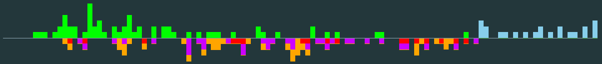


Problem: For each phone number, output the matching words.

Observation: $n \cdot m \leq 10^5$, so run-time complexity of $\mathcal{O}(nm)$ is fine.

D: Dialling Digits

Problem author: Ragnar Groot Koerkamp



Problem: For each phone number, output the matching words.

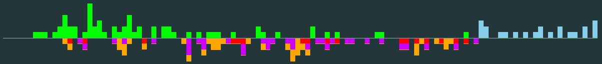
Observation: $n \cdot m \leq 10^5$, so run-time complexity of $\mathcal{O}(nm)$ is fine.

Solution: For each phone number p, for each word w:

- Let w_d be the letters in w converted to digits using the keypad.
- Add w the output of p if $w_d = p$.

D: Dialling Digits

Problem author: Ragnar Groot Koerkamp



Problem: For each phone number, output the matching words.

Observation: $n \cdot m \leq 10^5$, so run-time complexity of $\mathcal{O}(nm)$ is fine.

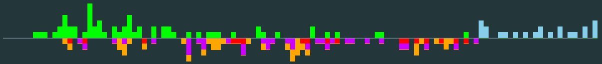
Solution: For each phone number p, for each word w:

- Let w_d be the letters in w converted to digits using the keypad.
- Add w the output of p if $w_d = p$.

Note: Can also be done in $\mathcal{O}(n + m)$ by precalculating the digits for each word.

D: Dialling Digits

Problem author: Ragnar Groot Koerkamp



Problem: For each phone number, output the matching words.

Observation: $n \cdot m \leq 10^5$, so run-time complexity of $\mathcal{O}(nm)$ is fine.

Solution: For each phone number p, for each word w:

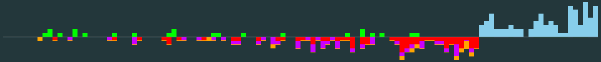
- Let w_d be the letters in w converted to digits using the keypad.
- Add w the output of p if $w_d = p$.

Note: Can also be done in $\mathcal{O}(n + m)$ by precalculating the digits for each word.

Statistics: 176 submissions, 63 accepted, 22 unknown

G: Giganotosaurus Game

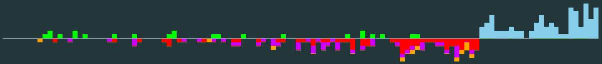
Problem author: Tobias Roehr



Problem: A game where you jump over cactuses, trying to reach the end of the world. Each jump is one cell longer than the last. How many different winning paths exist?

G: Giganotosaurus Game

Problem author: Tobias Roehr

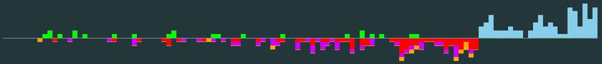


Problem: A game where you jump over cactuses, trying to reach the end of the world. Each jump is one cell longer than the last. How many different winning paths exist?

Observation: If you jump k times, you move past $1 + 2 + \ldots + k \in \mathcal{O}(k^2)$ cells. Hence, you can jump at most $\mathcal{O}(\sqrt{n})$ times.

G: Giganotosaurus Game

Problem author: Tobias Roehr



Problem: A game where you jump over cactuses, trying to reach the end of the world. Each jump is one cell longer than the last. How many different winning paths exist?

Observation: If you jump k times, you move past $1 + 2 + \ldots + k \in \mathcal{O}(k^2)$ cells. Hence, you can jump at most $\mathcal{O}(\sqrt{n})$ times.

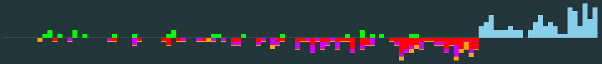
Solution: Let $A[x][k]$ denote the number of paths to cell x with exactly k jumps. You can reach this state by either jumping or not, so

$$A[x][k] = \begin{cases} 0 & \text{if there is a cactus at } x \\ A[x-k-1][k-1] + A[x-1][k] & \text{otherwise} \end{cases}$$

So we use *dynamic programming*. The answer is the sum of all values in A past n.

G: Giganotosaurus Game

Problem author: Tobias Roehr



Problem: A game where you jump over cactuses, trying to reach the end of the world. Each jump is one cell longer than the last. How many different winning paths exist?

Observation: If you jump k times, you move past $1 + 2 + \ldots + k \in \mathcal{O}(k^2)$ cells. Hence, you can jump at most $\mathcal{O}(\sqrt{n})$ times.

Solution: Let $A[x][k]$ denote the number of paths to cell x with exactly k jumps. You can reach this state by either jumping or not, so

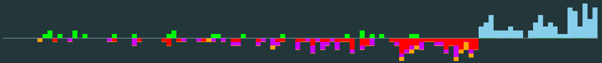
$$A[x][k] = \begin{cases} 0 & \text{if there is a cactus at } x \\ A[x-k-1][k-1] + A[x-1][k] & \text{otherwise} \end{cases}$$

So we use *dynamic programming*. The answer is the sum of all values in A past n.

Pitfall: Bounds checking in the recurrence. It is easier to use a *bottom-up* approach.

G: Giganotosaurus Game

Problem author: Tobias Roehr



Problem: A game where you jump over cactuses, trying to reach the end of the world. Each jump is one cell longer than the last. How many different winning paths exist?

Observation: If you jump k times, you move past $1 + 2 + \ldots + k \in \mathcal{O}(k^2)$ cells. Hence, you can jump at most $\mathcal{O}(\sqrt{n})$ times.

Solution: Let $A[x][k]$ denote the number of paths to cell x with exactly k jumps. You can reach this state by either jumping or not, so

$$A[x][k] = \begin{cases} 0 & \text{if there is a cactus at } x \\ A[x-k-1][k-1] + A[x-1][k] & \text{otherwise} \end{cases}$$

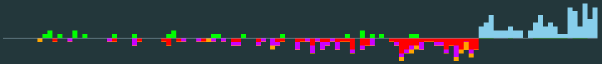
So we use *dynamic programming*. The answer is the sum of all values in A past n.

Pitfall: Bounds checking in the recurrence. It is easier to use a *bottom-up* approach.

Run time: $\mathcal{O}(n\sqrt{n})$, due to the size of the table.

G: Giganotosaurus Game

Problem author: Tobias Roehr



Problem: A game where you jump over cactuses, trying to reach the end of the world. Each jump is one cell longer than the last. How many different winning paths exist?

Observation: If you jump k times, you move past $1 + 2 + \ldots + k \in \mathcal{O}(k^2)$ cells. Hence, you can jump at most $\mathcal{O}(\sqrt{n})$ times.

Solution: Let $A[x][k]$ denote the number of paths to cell x with exactly k jumps. You can reach this state by either jumping or not, so

$$A[x][k] = \begin{cases} 0 & \text{if there is a cactus at } x \\ A[x-k-1][k-1] + A[x-1][k] & \text{otherwise} \end{cases}$$

So we use *dynamic programming*. The answer is the sum of all values in A past n.

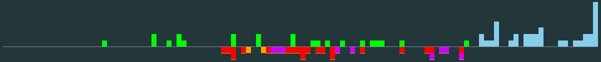
Pitfall: Bounds checking in the recurrence. It is easier to use a *bottom-up* approach.

Run time: $\mathcal{O}(n\sqrt{n})$, due to the size of the table.

Statistics: 232 submissions, 23 accepted, 90 unknown

L: Lawful Limits

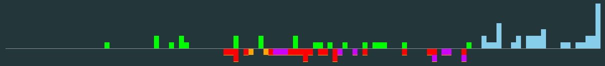
Problem author: Dirk van Bree



Problem: Find the length of the shortest path through a graph where the maximum speed of all edges increases at some time t.

L: Lawful Limits

Problem author: Dirk van Bree

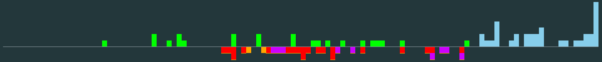


Problem: Find the length of the shortest path through a graph where the maximum speed of all edges increases at some time t.

Possible pitfall: If the speed limit increases when you are on a road, you can drive at that higher velocity instead.

L: Lawful Limits

Problem author: Dirk van Bree



Problem: Find the length of the shortest path through a graph where the maximum speed of all edges increases at some time t.

Possible pitfall: If the speed limit increases when you are on a road, you can drive at that higher velocity instead.

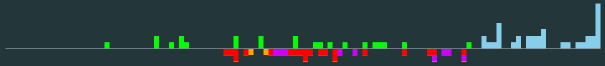
Remark: The time it takes to drive down a road of length ℓ with speeds $v_1 < v_2$ changing at time t is given by

$$\text{time} = \begin{cases} \ell/v_2 & T \geq t \\ \ell/v_1 & (t - T) \cdot v_1 > \ell \\ t - T + (\ell - (t - T) \cdot v_1)/v_2 & \text{else.} \end{cases}$$

when the current time is T.

L: Lawful Limits

Problem author: Dirk van Bree



Problem: Find the length of the shortest path through a graph where the maximum speed of all edges increases at some time t.

Possible pitfall: If the speed limit increases when you are on a road, you can drive at that higher velocity instead.

Remark: The time it takes to drive down a road of length ℓ with speeds $v_1 < v_2$ changing at time t is given by

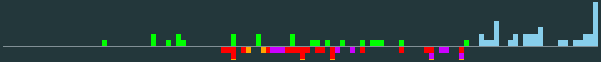
$$\text{time} = \begin{cases} \ell / v_2 & T \geq t \\ \ell / v_1 & (t - T) \cdot v_1 > \ell \\ t - T + (\ell - (t - T) \cdot v_1) / v_2 & \text{else.} \end{cases}$$

when the current time is T.

Solution: Apply Dijkstra to the time it takes to get to a vertex.

L: Lawful Limits

Problem author: Dirk van Bree



Problem: Find the length of the shortest path through a graph where the maximum speed of all edges increases at some time t.

Possible pitfall: If the speed limit increases when you are on a road, you can drive at that higher velocity instead.

Remark: The time it takes to drive down a road of length ℓ with speeds $v_1 < v_2$ changing at time t is given by

$$\text{time} = \begin{cases} \ell/v_2 & T \geq t \\ \ell/v_1 & (t - T) \cdot v_1 > \ell \\ t - T + (\ell - (t - T) \cdot v_1)/v_2 & \text{else.} \end{cases}$$

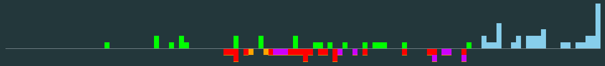
when the current time is T.

Solution: Apply Dijkstra to the time it takes to get to a vertex.

Run time: $\mathcal{O}(m + n \log n)$.

L: Lawful Limits

Problem author: Dirk van Bree



Problem: Find the length of the shortest path through a graph where the maximum speed of all edges increases at some time t.

Possible pitfall: If the speed limit increases when you are on a road, you can drive at that higher velocity instead.

Remark: The time it takes to drive down a road of length ℓ with speeds $v_1 < v_2$ changing at time t is given by

$$\text{time} = \begin{cases} \ell / v_2 & T \geq t \\ \ell / v_1 & (t - T) \cdot v_1 > \ell \\ t - T + (\ell - (t - T) \cdot v_1) / v_2 & \text{else.} \end{cases}$$

when the current time is T.

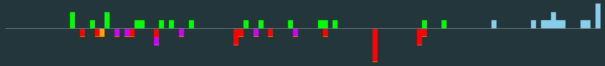
Solution: Apply Dijkstra to the time it takes to get to a vertex.

Run time: $\mathcal{O}(m + n \log n)$.

Statistics: 90 submissions, 23 accepted, 35 unknown

A: Awkward Auction

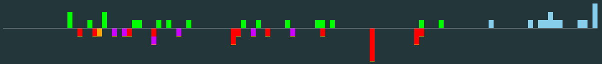
Problem author: Mees de Vries



Problem: Consider guessing a secret number m between 1 and n with feedback 'lower', 'higher', or 'correct'. Guessing $g < m$ has cost b, while guessing $g \geq m$ has cost g. Find the worst-case cost until guessing m, assuming you play optimally.

A: Awkward Auction

Problem author: Mees de Vries

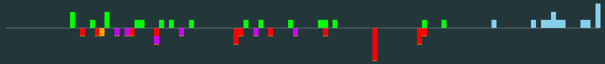


Problem: Consider guessing a secret number m between 1 and n with feedback 'lower', 'higher', or 'correct'. Guessing $g < m$ has cost b, while guessing $g \geq m$ has cost g. Find the worst-case cost until guessing m, assuming you play optimally.

Solution: Dynamic Programming.

A: Awkward Auction

Problem author: Mees de Vries



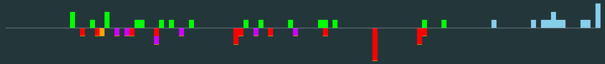
Problem: Consider guessing a secret number m between 1 and n with feedback 'lower', 'higher', or 'correct'. Guessing $g < m$ has cost b, while guessing $g \geq m$ has cost g. Find the worst-case cost until guessing m, assuming you play optimally.

Solution: Dynamic Programming.

- For all $1 \leq x \leq y \leq n$, find the optimal worst-case cost dp$[x][y]$ of guessing a number in the interval $[x, y]$.

A: Awkward Auction

Problem author: Mees de Vries



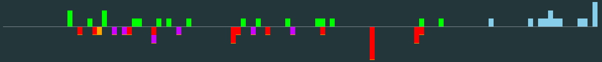
Problem: Consider guessing a secret number m between 1 and n with feedback ‘lower’, ‘higher’, or ‘correct’. Guessing $g < m$ has cost b, while guessing $g \geq m$ has cost g. Find the worst-case cost until guessing m, assuming you play optimally.

Solution: Dynamic Programming.

- For all $1 \leq x \leq y \leq n$, find the optimal worst-case cost $\mathrm{dp}[x][y]$ of guessing a number in the interval $[x, y]$.
- Compute the $\mathrm{dp}[x][y]$ in increasing order of the length $y - x$ of the interval.

A: Awkward Auction

Problem author: Mees de Vries



Problem: Consider guessing a secret number m between 1 and n with feedback 'lower', 'higher', or 'correct'. Guessing $g < m$ has cost b, while guessing $g \geq m$ has cost g. Find the worst-case cost until guessing m, assuming you play optimally.

Solution: Dynamic Programming.

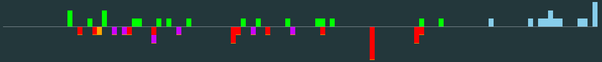
- For all $1 \leq x \leq y \leq n$, find the optimal worst-case cost $\mathrm{dp}[x][y]$ of guessing a number in the interval $[x, y]$.
- Compute the $\mathrm{dp}[x][y]$ in increasing order of the length $y - x$ of the interval.
- Then $\mathrm{dp}[x][x] = x$ (since we have to guess x), $\mathrm{dp}[x][y] = 0$ if $x > y$ and

$$\mathrm{dp}[x][y] = \min\left\{ \min_{x \leq g < y} \left[\max\left\{ \underbrace{g + \mathrm{dp}[x][g-1]}_{\text{guess too high}}, \underbrace{b + \mathrm{dp}[g+1][y]}_{\text{guess too low}} \right\} \right], \underbrace{y + \mathrm{dp}[x][y-1]}_{\text{guess too high}} \right\}.$$

Guessing right is always cheaper than guessing too high, so we can leave it out.

A: Awkward Auction

Problem author: Mees de Vries



Problem: Consider guessing a secret number m between 1 and n with feedback ‘lower’, ‘higher’, or ‘correct’. Guessing $g < m$ has cost b, while guessing $g \geq m$ has cost g. Find the worst-case cost until guessing m, assuming you play optimally.

Solution: Dynamic Programming.

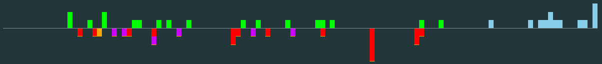
- For all $1 \leq x \leq y \leq n$, find the optimal worst-case cost $\mathrm{dp}[x][y]$ of guessing a number in the interval $[x, y]$.
- Compute the $\mathrm{dp}[x][y]$ in increasing order of the length $y - x$ of the interval.
- Then $\mathrm{dp}[x][x] = x$ (since we have to guess x), $\mathrm{dp}[x][y] = 0$ if $x > y$ and

$$\mathrm{dp}[x][y] = \min \left\{ \min_{x \leq g < y} \left[\max \left\{ \underbrace{g + \mathrm{dp}[x][g-1]}_{\text{guess too high}}, \underbrace{b + \mathrm{dp}[g+1][y]}_{\text{guess too low}} \right\} \right], \underbrace{y + \mathrm{dp}[x][y-1]}_{\text{guess too high}} \right\}.$$

  Guessing right is always cheaper than guessing too high, so we can leave it out.
- The answer is $\mathrm{dp}[1][n]$.

A: Awkward Auction

Problem author: Mees de Vries



Problem: Consider guessing a secret number m between 1 and n with feedback 'lower', 'higher', or 'correct'. Guessing $g < m$ has cost b, while guessing $g \geq m$ has cost g. Find the worst-case cost until guessing m, assuming you play optimally.

Solution: Dynamic Programming.

- For all $1 \leq x \leq y \leq n$, find the optimal worst-case cost $\mathrm{dp}[x][y]$ of guessing a number in the interval $[x, y]$.
- Compute the $\mathrm{dp}[x][y]$ in increasing order of the length $y - x$ of the interval.
- Then $\mathrm{dp}[x][x] = x$ (since we have to guess x), $\mathrm{dp}[x][y] = 0$ if $x > y$ and

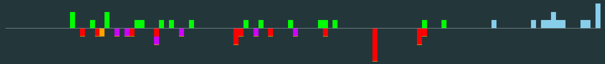
$$\mathrm{dp}[x][y] = \min \left\{ \min_{x \leq g < y} \left[\max \left\{ \underbrace{g + \mathrm{dp}[x][g-1]}_{\text{guess too high}}, \underbrace{b + \mathrm{dp}[g+1][y]}_{\text{guess too low}} \right\} \right], \underbrace{y + \mathrm{dp}[x][y-1]}_{\text{guess too high}} \right\}.$$

  Guessing right is always cheaper than guessing too high, so we can leave it out.
- The answer is $\mathrm{dp}[1][n]$.

Run time: $\mathcal{O}(n^3)$.

A: Awkward Auction

Problem author: Mees de Vries



Problem: Consider guessing a secret number m between 1 and n with feedback 'lower', 'higher', or 'correct'. Guessing $g < m$ has cost b, while guessing $g \geq m$ has cost g. Find the worst-case cost until guessing m, assuming you play optimally.

Solution: Dynamic Programming.

- For all $1 \leq x \leq y \leq n$, find the optimal worst-case cost $\mathrm{dp}[x][y]$ of guessing a number in the interval $[x, y]$.
- Compute the $\mathrm{dp}[x][y]$ in increasing order of the length $y - x$ of the interval.
- Then $\mathrm{dp}[x][x] = x$ (since we have to guess x), $\mathrm{dp}[x][y] = 0$ if $x > y$ and

$$\mathrm{dp}[x][y] = \min \left\{ \min_{x \leq g < y} \left[\max \left\{ \underbrace{g + \mathrm{dp}[x][g-1]}_{\text{guess too high}}, \underbrace{b + \mathrm{dp}[g+1][y]}_{\text{guess too low}} \right\} \right], \underbrace{y + \mathrm{dp}[x][y-1]}_{\text{guess too high}} \right\}.$$

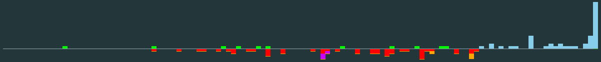
  Guessing right is always cheaper than guessing too high, so we can leave it out.
- The answer is $\mathrm{dp}[1][n]$.

Run time: $\mathcal{O}(n^3)$.

Statistics: 54 submissions, 18 accepted, 13 unknown

F: Fractal Area

Problem author: Lammert Westerdijk



Problem: Determine the area of a triangle, where the edges are a fractal defined by a polyline.

F: Fractal Area

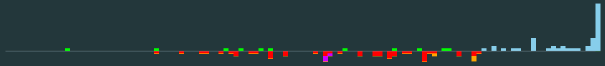
Problem author: Lammert Westerdijk

Problem: Determine the area of a triangle, where the edges are a fractal defined by a polyline.

Solution base: The area of the equilateral triangle with sides of length 1 is $\sqrt{3}/4$.

F: Fractal Area

Problem author: Lammert Westerdijk



Problem: Determine the area of a triangle, where the edges are a fractal defined by a polyline.

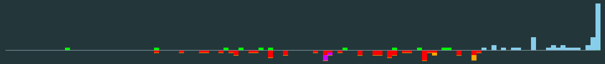
Solution base: The area of the equilateral triangle with sides of length 1 is $\sqrt{3}/4$.

One step: The area below the polyline can be calculated using the trapezoidal rule:

$$\sum_{i=1}^{n-1} (x_{i+1} - x_i) \cdot \frac{1}{2}(y_i + y_{i+1})$$

F: Fractal Area

Problem author: Lammert Westerdijk



Problem: Determine the area of a triangle, where the edges are a fractal defined by a polyline.

Solution base: The area of the equilateral triangle with sides of length 1 is $\sqrt{3}/4$.

One step: The area below the polyline can be calculated using the trapezoidal rule:

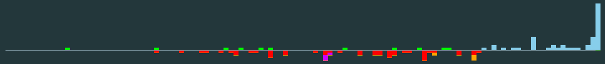
$$\sum_{i=1}^{n-1} (x_{i+1} - x_i) \cdot \frac{1}{2}(y_i + y_{i+1})$$

Next step: If the area of level k of the fractal is A_k, the area of the next level is multiplied by the square of lengths of the line segments:

$$A_{k+1} = \sum_{i=1}^{n-1} d(i, i+1)^2 A_k \qquad \big(d(i,j) \equiv \text{distance between points } i \text{ and } j\big)$$

F: Fractal Area

Problem author: Lammert Westerdijk



Problem: Determine the area of a triangle, where the edges are a fractal defined by a polyline.

Solution base: The area of the equilateral triangle with sides of length 1 is $\sqrt{3}/4$.

One step: The area below the polyline can be calculated using the trapezoidal rule:

$$\sum_{i=1}^{n-1} (x_{i+1} - x_i) \cdot \frac{1}{2}(y_i + y_{i+1})$$

Next step: If the area of level k of the fractal is A_k, the area of the next level is multiplied by the square of lengths of the line segments:

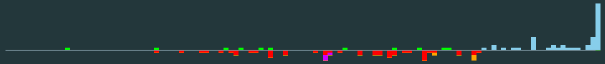
$$A_{k+1} = \sum_{i=1}^{n-1} d(i, i+1)^2 A_k \qquad \big(d(i,j) \equiv \text{distance between points } i \text{ and } j\big)$$

Final answer: Sum areas of all levels and multiply by the 3 sides:

$$\frac{\sqrt{3}}{4} + 3\sum_{k=0}^{\infty} A_k$$

F: Fractal Area

Problem author: Lammert Westerdijk



Problem: Determine the area of a triangle, where the edges are a fractal defined by a polyline.

Solution base: The area of the equilateral triangle with sides of length 1 is $\sqrt{3}/4$.

One step: The area below the polyline can be calculated using the trapezoidal rule:

$$\sum_{i=1}^{n-1}(x_{i+1} - x_i) \cdot \frac{1}{2}(y_i + y_{i+1})$$

Next step: If the area of level k of the fractal is A_k, the area of the next level is multiplied by the square of lengths of the line segments:

$$A_{k+1} = \sum_{i=1}^{n-1} d(i, i+1)^2 A_k \qquad \big(d(i,j) \equiv \text{distance between points } i \text{ and } j\big)$$

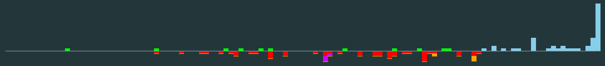
Final answer: Sum areas of all levels and multiply by the 3 sides:

$$\frac{\sqrt{3}}{4} + 3\sum_{k=0}^{\infty} A_k$$

But summing to ∞ is difficult... [*citation needed*]

F: Fractal Area

Problem author: Lammert Westerdijk

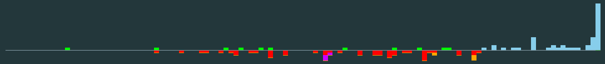


Problem: Determine the area of a triangle, where the edges are a fractal defined by a polyline.

Problem 2: Calculate $\frac{\sqrt{3}}{4} + 3\sum_{k=0}^{\infty} A_k$ without actually summing to ∞.

F: Fractal Area

Problem author: Lammert Westerdijk



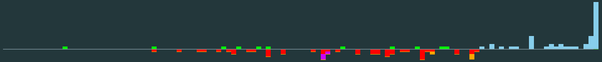
Problem: Determine the area of a triangle, where the edges are a fractal defined by a polyline.

Problem 2: Calculate $\dfrac{\sqrt{3}}{4} + 3\displaystyle\sum_{k=0}^{\infty} A_k$ without actually summing to ∞.

Solve recurrence: Write A_k as $r^k \cdot A_0$ (r is the constant ratio of areas between two levels).
The sum of a geometric series is $\sum_{k=0}^{\infty} r^k \cdot A_0 = \frac{A_0}{1-r}$.

F: Fractal Area

Problem author: Lammert Westerdijk



Problem: Determine the area of a triangle, where the edges are a fractal defined by a polyline.

Problem 2: Calculate $\frac{\sqrt{3}}{4} + 3\sum_{k=0}^{\infty} A_k$ without actually summing to ∞.

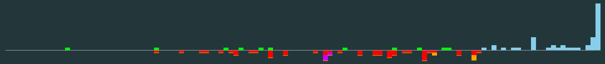
Solve recurrence: Write A_k as $r^k \cdot A_0$ (r is the constant ratio of areas between two levels).
The sum of a geometric series is $\sum_{k=0}^{\infty} r^k \cdot A_0 = \frac{A_0}{1-r}$.

Final answer v2.0:

$$\frac{\sqrt{3}}{4} + 3 \cdot \frac{A_0}{1-r}$$

F: Fractal Area

Problem author: Lammert Westerdijk



Problem: Determine the area of a triangle, where the edges are a fractal defined by a polyline.

Problem 2: Calculate $\frac{\sqrt{3}}{4} + 3\sum_{k=0}^{\infty} A_k$ without actually summing to ∞.

Solve recurrence: Write A_k as $r^k \cdot A_0$ (r is the constant ratio of areas between two levels).
The sum of a geometric series is $\sum_{k=0}^{\infty} r^k \cdot A_0 = \frac{A_0}{1-r}$.

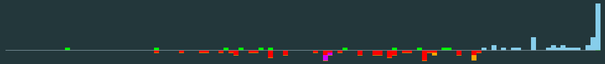
Final answer v2.0:

$$\frac{\sqrt{3}}{4} + 3 \cdot \frac{A_0}{1-r}$$

Run time: $\mathcal{O}(n)$ to calculate A_0 (area below polyline) and r (sum of squares of segment lengths).

F: Fractal Area

Problem author: Lammert Westerdijk



Problem: Determine the area of a triangle, where the edges are a fractal defined by a polyline.

Problem 2: Calculate $\frac{\sqrt{3}}{4} + 3\sum_{k=0}^{\infty} A_k$ without actually summing to ∞.

Solve recurrence: Write A_k as $r^k \cdot A_0$ (r is the constant ratio of areas between two levels).
The sum of a geometric series is $\sum_{k=0}^{\infty} r^k \cdot A_0 = \frac{A_0}{1-r}$.

Final answer v2.0:

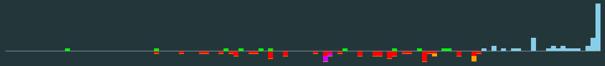
$$\frac{\sqrt{3}}{4} + 3 \cdot \frac{A_0}{1-r}$$

Run time: $\mathcal{O}(n)$ to calculate A_0 (area below polyline) and r (sum of squares of segment lengths).

But. . . : A 64-bit `double` is not infinite! Looping and summing until the answer does not change anymore is possible, this terminates after a few million iterations.

F: Fractal Area

Problem author: Lammert Westerdijk



Problem: Determine the area of a triangle, where the edges are a fractal defined by a polyline.

Problem 2: Calculate $\frac{\sqrt{3}}{4} + 3\sum_{k=0}^{\infty} A_k$ without actually summing to ∞.

Solve recurrence: Write A_k as $r^k \cdot A_0$ (r is the constant ratio of areas between two levels).
The sum of a geometric series is $\sum_{k=0}^{\infty} r^k \cdot A_0 = \frac{A_0}{1-r}$.

Final answer v2.0:

$$\frac{\sqrt{3}}{4} + 3 \cdot \frac{A_0}{1-r}$$

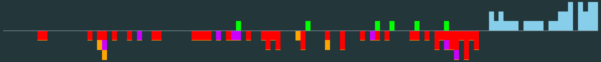
Run time: $\mathcal{O}(n)$ to calculate A_0 (area below polyline) and r (sum of squares of segment lengths).

But. . . : A 64-bit `double` is not infinite! Looping and summing until the answer does not change anymore is possible, this terminates after a few million iterations.

Statistics: 106 submissions, 11 accepted, 45 unknown

C: Chaotic Cables

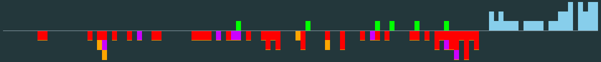
Problem author: Mees de Vries



Problem: Recognize a hypercube.

C: Chaotic Cables

Problem author: Mees de Vries

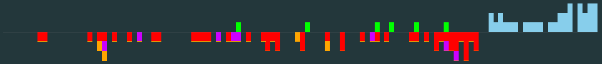


Problem: Recognize a hypercube.

Observation: There are 2^d vertices in an d-dimensional hypercube and each vertex is connected to exactly d other vertices.

C: Chaotic Cables

Problem author: Mees de Vries



Problem: Recognize a hypercube.

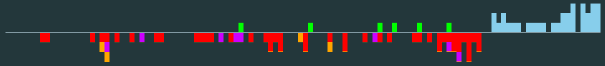
Observation: There are 2^d vertices in an d-dimensional hypercube and each vertex is connected to exactly d other vertices.

Solution:

- Pick an arbitrary vertex v and label it as 0.
- Label all neighbours of v with distinct powers of 2.
- Do a breadth-first search from v. For each unvisited neighbour u of v, label u with the bitwise OR of its current label and the label of v.
- Check for each edge if the labels of its endpoints differ in exactly one bit.

C: Chaotic Cables

Problem author: Mees de Vries



Problem: Recognize a hypercube.

Observation: There are 2^d vertices in an d-dimensional hypercube and each vertex is connected to exactly d other vertices.

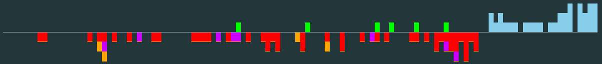
Solution:

- Pick an arbitrary vertex v and label it as 0.
- Label all neighbours of v with distinct powers of 2.
- Do a breadth-first search from v. For each unvisited neighbour u of v, label u with the bitwise OR of its current label and the label of v.
- Check for each edge if the labels of its endpoints differ in exactly one bit.

Pitfall: Missing checks or checking the number of edges instead of each vertex degree may lead to wrong answers.

C: Chaotic Cables

Problem author: Mees de Vries



Problem: Recognize a hypercube.

Observation: There are 2^d vertices in an d-dimensional hypercube and each vertex is connected to exactly d other vertices.

Solution:

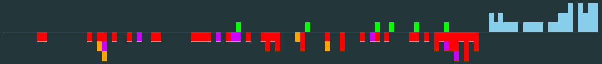
- Pick an arbitrary vertex v and label it as 0.
- Label all neighbours of v with distinct powers of 2.
- Do a breadth-first search from v. For each unvisited neighbour u of v, label u with the bitwise OR of its current label and the label of v.
- Check for each edge if the labels of its endpoints differ in exactly one bit.

Pitfall: Missing checks or checking the number of edges instead of each vertex degree may lead to wrong answers.

Run time: $\mathcal{O}(n + m)$.

C: Chaotic Cables

Problem author: Mees de Vries



Problem: Recognize a hypercube.

Observation: There are 2^d vertices in an d-dimensional hypercube and each vertex is connected to exactly d other vertices.

Solution:

- Pick an arbitrary vertex v and label it as 0.
- Label all neighbours of v with distinct powers of 2.
- Do a breadth-first search from v. For each unvisited neighbour u of v, label u with the bitwise OR of its current label and the label of v.
- Check for each edge if the labels of its endpoints differ in exactly one bit.

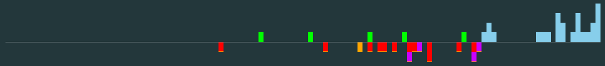
Pitfall: Missing checks or checking the number of edges instead of each vertex degree may lead to wrong answers.

Run time: $\mathcal{O}(n + m)$.

Statistics: 98 submissions, 6 accepted, 32 unknown

I: Investment Investigation

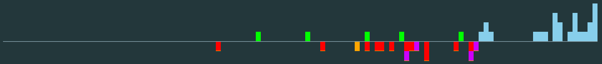
Problem author: Ivan Fefer



Problem: Given a list of all orders made on a stock market, generate a list of all transactions made. Normal orders can be fulfilled after being placed, while FoK orders need to be fulfilled instantaneously or not at all.

I: Investment Investigation

Problem author: Ivan Fefer

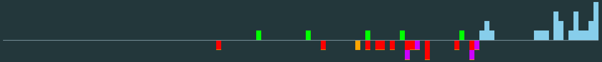


Problem: Given a list of all orders made on a stock market, generate a list of all transactions made. Normal orders can be fulfilled after being placed, while FoK orders need to be fulfilled instantaneously or not at all.

Observation: Every transaction completes at least one order, so the number of transactions is $\mathcal{O}(n)$.

I: Investment Investigation

Problem author: Ivan Fefer



Problem: Given a list of all orders made on a stock market, generate a list of all transactions made. Normal orders can be fulfilled after being placed, while FoK orders need to be fulfilled instantaneously or not at all.

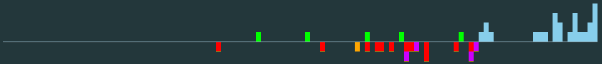
Observation: Every transaction completes at least one order, so the number of transactions is $\mathcal{O}(n)$.

Observation: Normal orders can be handled with a priority queue.

- Run time: $\mathcal{O}(\#\text{transactions} \cdot \log n)$ per order.

I: Investment Investigation

Problem author: Ivan Fefer



Problem: Given a list of all orders made on a stock market, generate a list of all transactions made. Normal orders can be fulfilled after being placed, while FoK orders need to be fulfilled instantaneously or not at all.

Observation: Every transaction completes at least one order, so the number of transactions is $\mathcal{O}(n)$.

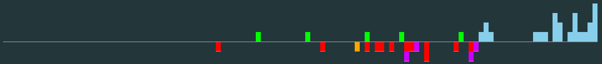
Observation: Normal orders can be handled with a priority queue.

- Run time: $\mathcal{O}(\#\text{transactions} \cdot \log n)$ per order.

Naive solution: Try handling a FoK order the same as a normal order, undoing transactions if it is not fulfilled.

I: Investment Investigation

Problem author: Ivan Fefer



Problem: Given a list of all orders made on a stock market, generate a list of all transactions made. Normal orders can be fulfilled after being placed, while FoK orders need to be fulfilled instantaneously or not at all.

Observation: Every transaction completes at least one order, so the number of transactions is $\mathcal{O}(n)$.

Observation: Normal orders can be handled with a priority queue.

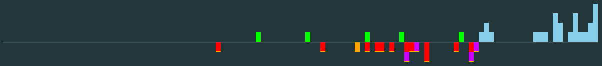
- Run time: $\mathcal{O}(\#\text{transactions} \cdot \log n)$ per order.

Naive solution: Try handling a FoK order the same as a normal order, undoing transactions if it is not fulfilled.

Problem: Can take $\mathcal{O}(n \log n)$ time per FoK order!

I: Investment Investigation

Problem author: Ivan Fefer



Problem: Given a list of all orders made on a stock market, generate a list of all transactions made. Normal orders can be fulfilled after being placed, while FoK orders need to be fulfilled instantaneously or not at all.

Observation: Every transaction completes at least one order, so the number of transactions is $\mathcal{O}(n)$.

Observation: Normal orders can be handled with a priority queue.

- Run time: $\mathcal{O}(\#\text{transactions} \cdot \log n)$ per order.

Naive solution: Try handling a FoK order the same as a normal order, undoing transactions if it is not fulfilled.

Problem: Can take $\mathcal{O}(n \log n)$ time per FoK order!

Observation: Need to quickly check whether a FoK order can be fulfilled.

I: Investment Investigation

Problem author: Ivan Fefer

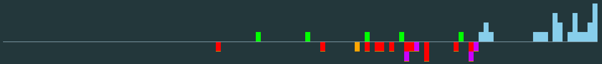
Problem: Given a list of all orders made on a stock market, generate a list of all transactions made. Normal orders can be fulfilled after being placed, while FoK orders need to be fulfilled instantaneously or not at all.

Observation: Normal orders can be handled with a priority queue.

Observation: Need to quickly check whether a FoK order can be fulfilled.

I: Investment Investigation

Problem author: Ivan Fefer



Problem: Given a list of all orders made on a stock market, generate a list of all transactions made. Normal orders can be fulfilled after being placed, while FoK orders need to be fulfilled instantaneously or not at all.

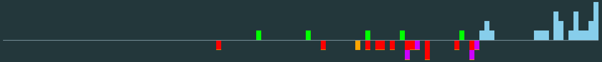
Observation: Normal orders can be handled with a priority queue.

Observation: Need to quickly check whether a FoK order can be fulfilled.

Online Solution: Use augmented binary search tree or implicit segment tree to compute total volume of outstanding orders above a buy price / below a sell price in $\mathcal{O}(\log n)$.

I: Investment Investigation

Problem author: Ivan Fefer



Problem: Given a list of all orders made on a stock market, generate a list of all transactions made. Normal orders can be fulfilled after being placed, while FoK orders need to be fulfilled instantaneously or not at all.

Observation: Normal orders can be handled with a priority queue.

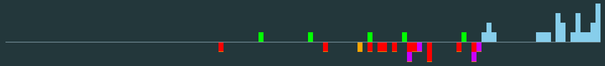
Observation: Need to quickly check whether a FoK order can be fulfilled.

Online Solution: Use augmented binary search tree or implicit segment tree to compute total volume of outstanding orders above a buy price / below a sell price in $\mathcal{O}(\log n)$.

- An augmented binary search tree can be difficult to implement.

I: Investment Investigation

Problem author: Ivan Fefer



Problem: Given a list of all orders made on a stock market, generate a list of all transactions made. Normal orders can be fulfilled after being placed, while FoK orders need to be fulfilled instantaneously or not at all.

Observation: Normal orders can be handled with a priority queue.

Observation: Need to quickly check whether a FoK order can be fulfilled.

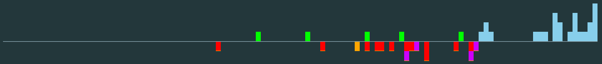
Online Solution: Use augmented binary search tree or implicit segment tree to compute total volume of outstanding orders above a buy price / below a sell price in $\mathcal{O}(\log n)$.

- An augmented binary search tree can be difficult to implement.

Offline Solution: Use a normal segment tree or binary indexed tree.

I: Investment Investigation

Problem author: Ivan Fefer



Problem: Given a list of all orders made on a stock market, generate a list of all transactions made. Normal orders can be fulfilled after being placed, while FoK orders need to be fulfilled instantaneously or not at all.

Observation: Normal orders can be handled with a priority queue.

Observation: Need to quickly check whether a FoK order can be fulfilled.

Online Solution: Use augmented binary search tree or implicit segment tree to compute total volume of outstanding orders above a buy price / below a sell price in $\mathcal{O}(\log n)$.

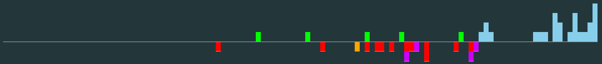
- An augmented binary search tree can be difficult to implement.

Offline Solution: Use a normal segment tree or binary indexed tree.

- Need to convert prices to values between 1 and n.

I: Investment Investigation

Problem author: Ivan Fefer



Problem: Given a list of all orders made on a stock market, generate a list of all transactions made. Normal orders can be fulfilled after being placed, while FoK orders need to be fulfilled instantaneously or not at all.

Observation: Normal orders can be handled with a priority queue.

Observation: Need to quickly check whether a FoK order can be fulfilled.

Online Solution: Use augmented binary search tree or implicit segment tree to compute total volume of outstanding orders above a buy price / below a sell price in $\mathcal{O}(\log n)$.

- An augmented binary search tree can be difficult to implement.

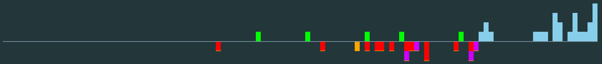
Offline Solution: Use a normal segment tree or binary indexed tree.

- Need to convert prices to values between 1 and n.

Run time: $\mathcal{O}(n \log n)$.

I: Investment Investigation

Problem author: Ivan Fefer



Problem: Given a list of all orders made on a stock market, generate a list of all transactions made. Normal orders can be fulfilled after being placed, while FoK orders need to be fulfilled instantaneously or not at all.

Observation: Normal orders can be handled with a priority queue.

Observation: Need to quickly check whether a FoK order can be fulfilled.

Online Solution: Use augmented binary search tree or implicit segment tree to compute total volume of outstanding orders above a buy price / below a sell price in $\mathcal{O}(\log n)$.

- An augmented binary search tree can be difficult to implement.

Offline Solution: Use a normal segment tree or binary indexed tree.

- Need to convert prices to values between 1 and n.

Run time: $\mathcal{O}(n \log n)$.

Statistics: 46 submissions, 5 accepted, 24 unknown

J: Joppiesaus Jailbreak

Problem author: Mike de Vries

Problem: Given the lengths $x_1, \ldots, x_n$ of all levels in a platformer, all of which take an integer number of frames to finish, determine the fastest time to finish all levels if the framerate can be set to any real in $(0, f]$.

J: Joppiesaus Jailbreak

Problem author: Mike de Vries

Problem: Given the lengths $x_1, \ldots, x_n$ of all levels in a platformer, all of which take an integer number of frames to finish, determine the fastest time to finish all levels if the framerate can be set to any real in $(0, f]$.

Reformulating: With framerate f', each level takes $\lceil x_i f'/1000 \rceil$ frames to finish. The total time is $(1/f') \sum_{i=1}^{n} \lceil x_i f'/1000 \rceil$.

J: Joppiesaus Jailbreak

Problem author: Mike de Vries

Problem: Given the lengths $x_1, \dots, x_n$ of all levels in a platformer, all of which take an integer number of frames to finish, determine the fastest time to finish all levels if the framerate can be set to any real in $(0, f]$.

Reformulating: With framerate f', each level takes $\lceil x_i f'/1000 \rceil$ frames to finish. The total time is $(1/f') \sum_{i=1}^{n} \lceil x_i f'/1000 \rceil$.

Observation 1: The function $1/f'$ is decreasing, so a minimum can only be attained when $\sum_{i=1}^{n} \lceil x_i f'/1000 \rceil$ jumps, **or when $\mathbf{f' = f}$**. Jumps occur whenever $f' = 1000m/x_i$ for some integer $0 < m \leq x_i f/1000$.

J: Joppiesaus Jailbreak

Problem author: Mike de Vries

Problem: Given the lengths $x_1, \ldots, x_n$ of all levels in a platformer, all of which take an integer number of frames to finish, determine the fastest time to finish all levels if the framerate can be set to any real in $(0, f]$.

Reformulating: With framerate f', each level takes $\lceil x_i f'/1000 \rceil$ frames to finish. The total time is $(1/f') \sum_{i=1}^{n} \lceil x_i f'/1000 \rceil$.

Observation 1: The function $1/f'$ is decreasing, so a minimum can only be attained when $\sum_{i=1}^{n} \lceil x_i f'/1000 \rceil$ jumps, **or when** $\mathbf{f' = f}$. Jumps occur whenever $f' = 1000m/x_i$ for some integer $0 < m \leq x_i f/1000$.

Naive solution: Compute all interesting framerates, and for each compute the total time to finish the game. This is $\mathcal{O}(nf \sum_{i=1}^{n} x_i/1000)$, too slow!

J: Joppiesaus Jailbreak

Problem author: Mike de Vries

Problem: Given the lengths $x_1, \ldots, x_n$ of all levels in a platformer, all of which take an integer number of frames to finish, determine the fastest time to finish all levels if the framerate can be set to any real in $(0, f]$.

Observation 2: If all jumps are distinct, the total number of frames increases by exactly 1 at each jump. If we sort the jumps, recomputing the total time takes $\mathcal{O}(1)$! This also works if the jumps are not distinct.

J: Joppiesaus Jailbreak

Problem author: Mike de Vries

Problem: Given the lengths $x_1, \ldots, x_n$ of all levels in a platformer, all of which take an integer number of frames to finish, determine the fastest time to finish all levels if the framerate can be set to any real in $(0, f]$.

Observation 2: If all jumps are distinct, the total number of frames increases by exactly 1 at each jump. If we sort the jumps, recomputing the total time takes $\mathcal{O}(1)$! This also works if the jumps are not distinct.

Solution: Compute all jumps and sort them. For the first jump, compute the total frames. For each jump after the first, simply add a single additional frame. Finally, compute the case $f' = f$.

J: Joppiesaus Jailbreak

Problem author: Mike de Vries

Problem: Given the lengths $x_1, \ldots, x_n$ of all levels in a platformer, all of which take an integer number of frames to finish, determine the fastest time to finish all levels if the framerate can be set to any real in $(0, f]$.

Observation 2: If all jumps are distinct, the total number of frames increases by exactly 1 at each jump. If we sort the jumps, recomputing the total time takes $\mathcal{O}(1)$! This also works if the jumps are not distinct.

Solution: Compute all jumps and sort them. For the first jump, compute the total frames. For each jump after the first, simply add a single additional frame. Finally, compute the case $f' = f$.

Run time: $\mathcal{O}(f \sum_{i=1}^{n} x_i / 1000 \log(f \sum_{i=1}^{n} x_i / 1000))$.

J: Joppiesaus Jailbreak

Problem author: Mike de Vries

Problem: Given the lengths $x_1, \ldots, x_n$ of all levels in a platformer, all of which take an integer number of frames to finish, determine the fastest time to finish all levels if the framerate can be set to any real in $(0, f]$.

Observation 2: If all jumps are distinct, the total number of frames increases by exactly 1 at each jump. If we sort the jumps, recomputing the total time takes $\mathcal{O}(1)$! This also works if the jumps are not distinct.

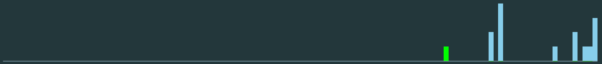
Solution: Compute all jumps and sort them. For the first jump, compute the total frames. For each jump after the first, simply add a single additional frame. Finally, compute the case $f' = f$.

Run time: $\mathcal{O}(f \sum_{i=1}^{n} x_i/1000 \log(f \sum_{i=1}^{n} x_i/1000))$.

Statistics: 47 submissions, 1 accepted, 39 unknown

K: Kitchens of Königsberg

Problem author: Tobias Roehr



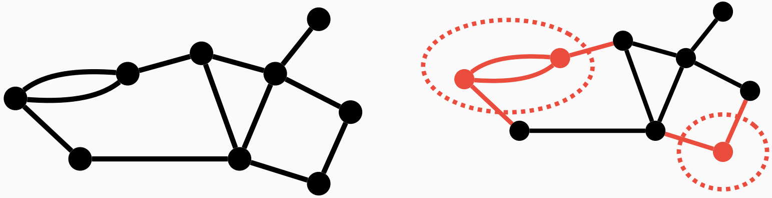
Problem: Given multigraph G, integer k. Find $K \subseteq V(G)$ such that exactly k edges have at least an endpoint in K. Also known as “Partial Exact Vertex Cover”.

K: Kitchens of Königsberg

Problem author: Tobias Roehr

Problem: Given multigraph G, integer k. Find $K \subseteq V(G)$ such that exactly k edges have at least an endpoint in K. Also known as "Partial Exact Vertex Cover".

Example: for $k = 6$:

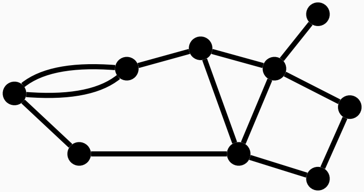


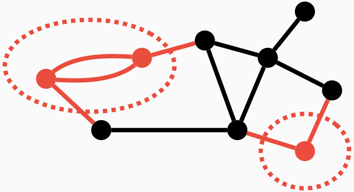
K: Kitchens of Königsberg

Problem author: Tobias Roehr

Problem: Given multigraph G, integer k. Find $K \subseteq V(G)$ such that exactly k edges have at least an endpoint in K. Also known as “Partial Exact Vertex Cover”.

Example: for $k = 6$:





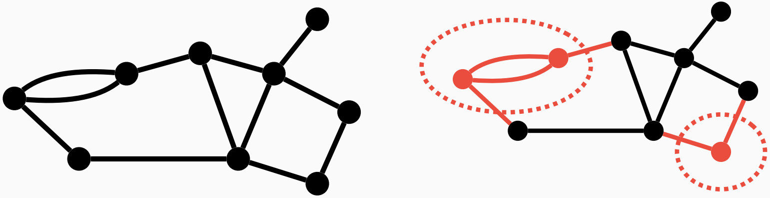
Naive solution 1: Consider all 2^n subsets of $V(G)$. Running time $\mathcal{O}(2^n \operatorname{poly}(n))$, way too slow.

K: Kitchens of Königsberg

Problem author: Tobias Roehr

Problem: Given multigraph G, integer k. Find $K \subseteq V(G)$ such that exactly k edges have at least an endpoint in K. Also known as "Partial Exact Vertex Cover".

Example: for $k = 6$:



Naive solution 1: Consider all 2^n subsets of $V(G)$. Running time $\mathcal{O}(2^n \operatorname{poly}(n))$, way too slow.

Naive solution 2: Can assume $|K| \leq k$, so it suffices to consider all $\binom{n}{1} + \cdots + \binom{n}{k} \leq n^k$ vertex subsets of size at most k. Running time $\mathcal{O}(n^k \operatorname{poly}(n))$, still too slow.

Problem: Given multigraph G, integer k. Find $K \subseteq V(G)$ such that exactly k edges have at least an endpoint in K. Also known as "Partial Exact Vertex Cover".

Hacky solution: The solution is very small ($|K| \leq k \leq 6$), so we can use preprocessing, exhaustive search, and local optimisation to solve what is otherwise an NP-hard problem even on large instances. Note that it must run in $\mathcal{O}(n^2)$.

Here are some ideas:

K: Kitchens of Königsberg

Problem author: Tobias Roehr

Problem: Given multigraph G, integer k. Find $K \subseteq V(G)$ such that exactly k edges have at least an endpoint in K. Also known as "Partial Exact Vertex Cover".

Hacky solution: The solution is very small ($|K| \leq k \leq 6$), so we can use preprocessing, exhaustive search, and local optimisation to solve what is otherwise an NP-hard problem even on large instances. Note that it must run in $\mathcal{O}(n^2)$.

Here are some ideas:

Remove large degree vertices: No vertex of degree $> k$ can contribute to the solution (it would cover $> k$ edges), so we can remove those from vertices under consideration for the vertex cover.

K: Kitchens of Königsberg

Problem author: Tobias Roehr

Problem: Given multigraph G, integer k. Find $K \subseteq V(G)$ such that exactly k edges have at least an endpoint in K. Also known as "Partial Exact Vertex Cover".

Hacky solution: The solution is very small ($|K| \leq k \leq 6$), so we can use preprocessing, exhaustive search, and local optimisation to solve what is otherwise an NP-hard problem even on large instances. Note that it must run in $\mathcal{O}(n^2)$.

Here are some ideas:

Remove large degree vertices: No vertex of degree $> k$ can contribute to the solution (it would cover $> k$ edges), so we can remove those from vertices under consideration for the vertex cover.

Check all singletons and pairs: The vertex set is small enough ($n \leq 5000$) that we can exhaustively check all $K \subseteq V(G)$ with $|K| \leq 2$.

K: Kitchens of Königsberg

Problem author: Tobias Roehr

Problem: Given multigraph G, integer k. Find $K \subseteq V(G)$ such that exactly k edges have at least an endpoint in K. Also known as "Partial Exact Vertex Cover".

Hacky solution: The solution is very small ($|K| \leq k \leq 6$), so we can use preprocessing, exhaustive search, and local optimisation to solve what is otherwise an NP-hard problem even on large instances. Note that it must run in $\mathcal{O}(n^2)$.

Here are some ideas:

Remove large degree vertices: No vertex of degree $> k$ can contribute to the solution (it would cover $> k$ edges), so we can remove those from vertices under consideration for the vertex cover.

Check all singletons and pairs: The vertex set is small enough ($n \leq 5000$) that we can exhaustively check all $K \subseteq V(G)$ with $|K| \leq 2$.

Remove leaves: For any leaf v, we can consider the instance $(G - \{v\}, k - 1)$ to detect every solution containing a leaf.

K: Kitchens of Königsberg

Problem author: Tobias Roehr

Problem: Given multigraph G, integer k. Find $K \subseteq V(G)$ such that exactly k edges have at least an endpoint in K. Also known as "Partial Exact Vertex Cover".

Hacky solution: The solution is very small ($|K| \leq k \leq 6$), so we can use preprocessing, exhaustive search, and local optimisation to solve what is otherwise an NP-hard problem even on large instances. Note that it must run in $\mathcal{O}(n^2)$.

Here are some ideas:

Remove large degree vertices: No vertex of degree $> k$ can contribute to the solution (it would cover $> k$ edges), so we can remove those from vertices under consideration for the vertex cover.

Check all singletons and pairs: The vertex set is small enough ($n \leq 5000$) that we can exhaustively check all $K \subseteq V(G)$ with $|K| \leq 2$.

Remove leaves: For any leaf v, we can consider the instance $(G - \{v\}, k - 1)$ to detect every solution containing a leaf.

Recursion: For any v, the graph (G, k) is a yes-instance if and only if $(G - \{v\}, k - \deg(v))$ is.

K: Kitchens of Königsberg

Problem author: Tobias Roehr

Problem: Given multigraph G, integer k. Find $K \subseteq V(G)$ such that exactly k edges have at least an endpoint in K. Also known as "Partial Exact Vertex Cover".

Hacky solution: The solution is very small ($|K| \leq k \leq 6$), so we can use preprocessing, exhaustive search, and local optimisation to solve what is otherwise an NP-hard problem even on large instances. Note that it must run in $\mathcal{O}(n^2)$.

Here are some ideas:

Remove large degree vertices: No vertex of degree $> k$ can contribute to the solution (it would cover $> k$ edges), so we can remove those from vertices under consideration for the vertex cover.

Check all singletons and pairs: The vertex set is small enough ($n \leq 5000$) that we can exhaustively check all $K \subseteq V(G)$ with $|K| \leq 2$.

Remove leaves: For any leaf v, we can consider the instance $(G - \{v\}, k - 1)$ to detect every solution containing a leaf.

Recursion: For any v, the graph (G, k) is a yes-instance if and only if $(G - \{v\}, k - \deg(v))$ is.

Fancy solution: Random orientation algorithm, see next slide.

Problem: Given multigraph G, integer k. Find $K \subseteq V(G)$ such that exactly k edges have at least an endpoint in K. Also known as "Partial Exact Vertex Cover".

Hacky solution: The solution is very small ($|K| \leq k \leq 6$), so we can use preprocessing, exhaustive search, and local optimisation to solve what is otherwise an NP-hard problem even on large instances. Note that it must run in $\mathcal{O}(n^2)$.

Here are some ideas:

Remove large degree vertices: No vertex of degree $> k$ can contribute to the solution (it would cover $> k$ edges), so we can remove those from vertices under consideration for the vertex cover.

Check all singletons and pairs: The vertex set is small enough ($n \leq 5000$) that we can exhaustively check all $K \subseteq V(G)$ with $|K| \leq 2$.

Remove leaves: For any leaf v, we can consider the instance $(G - \{v\}, k - 1)$ to detect every solution containing a leaf.

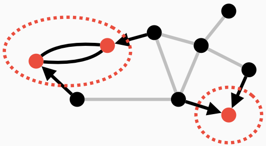
Recursion: For any v, the graph (G, k) is a yes-instance if and only if $(G - \{v\}, k - \deg(v))$ is.

Fancy solution: Random orientation algorithm, see next slide.

Statistics: 15 submissions, 1 accepted, 14 unknown

Problem: Given multigraph G, integer k. Find $K \subseteq V(G)$ such that exactly k edges have at least an endpoint in K. Also known as "Partial Exact Vertex Cover".

Random orientation algorithm. Randomly orient each edge uv as either (u, v), (v, u), or leave it undirected, each with probability $\frac{1}{3}$. Compute components $C_1, \ldots, C_r$ such that each C_i only contains arcs pointing *into* C_i. (Say, using BFS.)
Assemble solution from these C_i. ("Subset Sum" the indegrees of components to make k.)



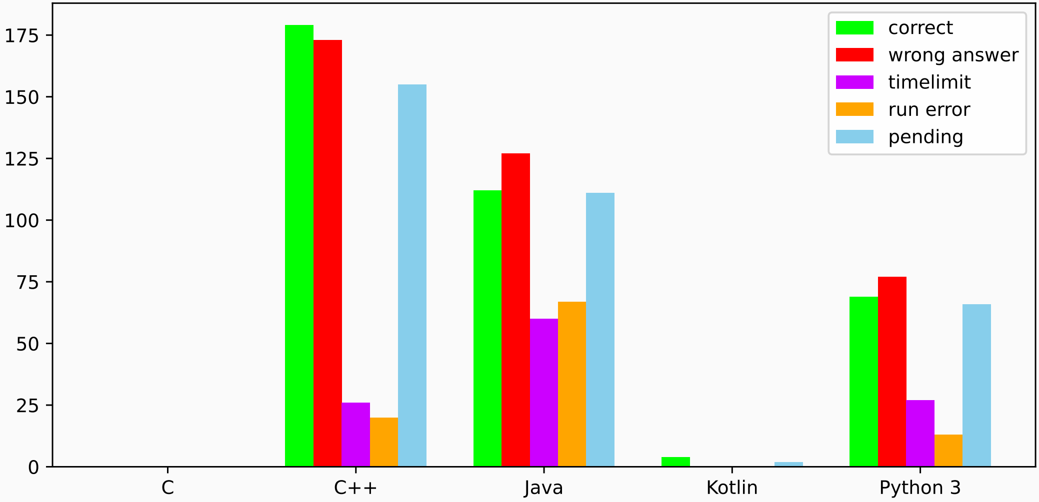
Correctness Every internal edge in K must remain undirected (probability $\frac{1}{3}$) and every edge incident on K must be directed towards (probability $\frac{1}{3}$). (Orientation of remaining edges unimportant.) Total success probability $= \frac{1}{3}^k$. Do $t = 3^k \ln n$ independent repetitions; all fail with probability

$$(1 - \tfrac{1}{3}^k)^t \leq (\exp(-\tfrac{1}{3}^k))^t \leq 1/n\,.$$

Run time $\mathcal{O}(3^k \operatorname{poly}(n))$, known as "fixed parameter tractable (FPT) in k".

[Kneis, J., Langer, A., Rossmanith, P. Improved Upper Bounds for Partial Vertex Cover. Graph-Theoretic Concepts in Computer Science. WG 2008. Springer LNCS 5344.]

Language stats



Random facts

Jury work

- 505 commits (last year: 492)

Random facts

Jury work

- 505 commits (last year: 492)
- 1228 secret test cases (last year: 1050) ($\approx 102\frac{1}{3}$ per problem!)

Random facts

Jury work

- 505 commits (last year: 492)
- 1228 secret test cases (last year: 1050) ($\approx 102\frac{1}{3}$ per problem!)
- 236 jury + proofreader solutions (last year: 195)

Random facts

Jury work

- 505 commits (last year: 492)
- 1228 secret test cases (last year: 1050) ($\approx 102\frac{1}{3}$ per problem!)
- 236 jury + proofreader solutions (last year: 195)
- The minimum[1] number of lines the jury needed to solve all problems is

$$4 + 3 + 7 + 3 + 2 + 3 + 21 + 1 + 60 + 21 + 61 + 9 = 195$$

  On average $16\frac{1}{4}$ lines per problem, up from 13.9 in last year's preliminaries

[1] With some code golfing

Thanks to:

The proofreaders

Angel Karchev
Arnoud van der Leer
Jaap Eldering
Jeroen Bransen (Java Hero 🎈)
Kevin Verbeek
Pavel Kunyavskiy (Kotlin Hero 🎈)
Thomas Verwoerd (Kotlin Hero 🎈)
Wendy Yi

The jury

Gijs Pennings
Jonas van der Schaaf
Jorke de Vlas
Lammert Westerdijk
Maarten Sijm
Mees de Vries
Mike de Vries
Ragnar Groot Koerkamp
Reinier Schmiermann
Thore Husfeldt
Tobias Roehr
Wietze Koops

Want to join the jury? Submit to the Call for Problems of BAPC 2025 at:

`https://jury.bapc.eu/`