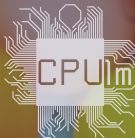




NWERC 2025

Solutions presentation

November 30, 2025

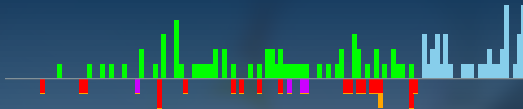


The NWERC 2025 Jury

- **Atli Fannar Franklín**
University of Iceland
- **Christopher Weyand**
Karlsruhe Institute of Technology / moia.io
- **Doan-Dai Nguyen**
École normale supérieure -
Université Paris Sciences &
Lettres
- **Jeroen Bransen**
Zympler
- **Jeroen Op de Beek**
Delft University of
Technology
- **Lucas Schwebler**
Karlsruhe Institute of
Technology
- **Maarten Sijm**
CHipCie (Delft University of
Technology) / lynk.so
- **Markus Himmel**
Lean FRO
- **Michael Zündorf**
Karlsruhe Institute of
Technology
- **Nils Gustafsson**
KTH Royal Institute of
Technology
- **Paul Wild**
FAU Erlangen-Nürnberg
- **Ragnar Groot Koerkamp**
- **Reinier Schmiermann**
University of Warsaw
- **Vitaly Aksenov**
City, University of London
- **Wendy Yi**
Karlsruhe Institute of
Technology
- **Yidi Zang**
Karlsruhe Institute of
Technology

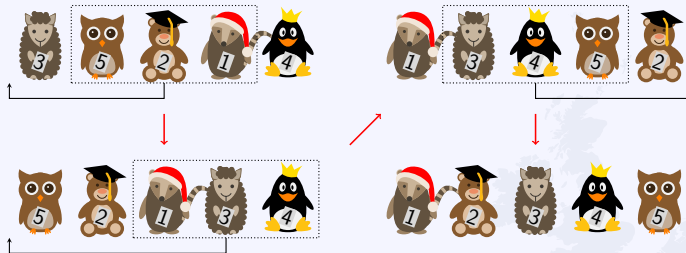
Big thanks to our proofreaders and test solvers

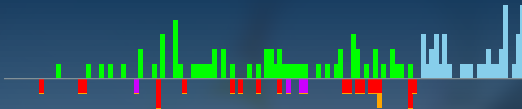
- **Andreas Grigorjew**
Paris Dauphine University - PSL
- **Arne Alex**
Ludwig-Maximilians-Universität München
- **Pavel Kunyavskiy**
JetBrains, Amsterdam
- **Vlad Ulmeanu**
Politehnica University of Bucharest



Problem

Given an unsorted sequence of the numbers $1, \dots, n$, sort it using operations of the following kind: cut three consecutive values from the sequence and paste them somewhere else.



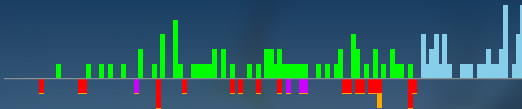


Problem

Given an unsorted sequence of the numbers $1, \dots, n$, sort it using operations of the following kind: cut three consecutive values from the sequence and paste them somewhere else.

Solution

- While there are more than 5 numbers, locate the largest one and move it to the back.
- This can be achieved in at most 2 steps and reduces the problem size by 1.
- For the last 5 numbers, either:
 - perform some graph search through the remaining $5! = 120$ permutations; or
 - just do random steps until the sequence is sorted.
- Total number of steps $\lesssim 2n + 160$



Problem

Given an unsorted sequence of the numbers $1, \dots, n$, sort it using operations of the following kind: cut three consecutive values from the sequence and paste them somewhere else.

Solution

- While there are more than 5 numbers, locate the largest one and move it to the back.
- This can be achieved in at most 2 steps and reduces the problem size by 1.
- For the last 5 numbers, either:
 - perform some graph search through the remaining $5! = 120$ permutations; or
 - just do random steps until the sequence is sorted.
- Total number of steps $\lesssim 2n + 160$

Problem

You want to withdraw $\text{€}n$ ($1 \leq n \leq 10\,000$) from a cash machine. It can dispense cash in the form of 1, 2, 5, 10, 20, 50, 100, 200, 500 (in euros). Check if the machine can dispense cash adding up to $\text{€}n$ which cannot be split evenly into two piles of $\text{€}n/2$.

Problem

You want to withdraw $\text{€}n$ ($1 \leq n \leq 10\,000$) from a cash machine. It can dispense cash in the form of 1, 2, 5, 10, 20, 50, 100, 200, 500 (in euros). Check if the machine can dispense cash adding up to $\text{€}n$ which cannot be split evenly into two piles of $\text{€}n/2$.

Naive solution (Subset Sum DP + Brute force)

Given a division into coins and notes summing to $\text{€}n$, classic subset sum dynamic programming can be used to check splittability:

$DP[\text{considering notes up to position } i][\text{sum of cash so far}] = \text{can choose subset with this sum}$

Because this DP uses True / False values, it can be sped up using bitsets, to achieve a complexity of $\mathcal{O}(\# \text{ notes } \frac{n}{w})$, where w is the wordsize, typically 32 or 64.

Problem

You want to withdraw $\text{€}n$ ($1 \leq n \leq 10\,000$) from a cash machine. It can dispense cash in the form of 1, 2, 5, 10, 20, 50, 100, 200, 500 (in euros). Check if the machine can dispense cash adding up to $\text{€}n$ which cannot be split evenly into two piles of $\text{€}n/2$.

Naive solution (Subset Sum DP + Brute force)

Given a division into coins and notes summing to $\text{€}n$, classic subset sum dynamic programming can be used to check splittability:

$DP[\text{considering notes up to position } i][\text{sum of cash so far}] = \text{can choose subset with this sum}$

Because this DP uses True / False values, it can be sped up using bitsets, to achieve a complexity of $\mathcal{O}(\# \text{ notes } \frac{n}{w})$, where w is the wordsize, typically 32 or 64.

Try out all possible splittings into all the different coins and notes, and check using the knapsack DP. This is too slow.

Problem

You want to withdraw $\text{€}n$ ($1 \leq n \leq 10\,000$) from a cash machine. It can dispense cash in the form of 1, 2, 5, 10, 20, 50, 100, 200, 500 (in euros). Check if the machine can dispense cash adding up to $\text{€}n$ which cannot be split evenly into two piles of $\text{€}n/2$.

Silly solution

Optimize the brute force as much as possible, using heuristics, and precompute all the answers on your own machine.

Then submit a big file which stores all possible answers up to 10 000 in it.

Observations

- For €1, €5, €10, €50 and €100, using 2 or more of the same coin or note is unnecessary.
For example, switching $€10 \times 2 \implies €20$ is strictly better.

Observations

- For €1, €5, €10, €50 and €100, using 2 or more of the same coin or note is unnecessary.
For example, switching $€10 \times 2 \implies €20$ is strictly better.
- For €2 and €20, using 5 or more of the same coin or note is unnecessary.
Switching $€2 \times 5 \implies €10$ is also strictly better.

Observations

- For €1, €5, €10, €50 and €100, using 2 or more of the same coin or note is unnecessary.
For example, switching $€10 \times 2 \implies €20$ is strictly better.
- For €2 and €20, using 5 or more of the same coin or note is unnecessary.
Switching $€2 \times 5 \implies €10$ is also strictly better.

Easy solution

Run the brute force, but use the above observations to reduce the number of options to check.

A rough upper bound for the number of options is $2^5 5^2 (10\,000/500) = 16\,000$, but in reality way less options actually sum to n . Here we brute force how many of the lower denominations we choose, also brute force the number of €500 notes.

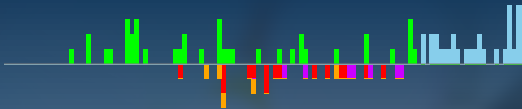
Greedy solution

It is almost possible to solve this problem greedily, but there are tricky corner cases. This is left as an exercise to the reader.

Greedy solution

It is almost possible to solve this problem greedily, but there are tricky corner cases. This is left as an exercise to the reader.

Statistics: 322 submissions, 33 accepted, 141 unknown

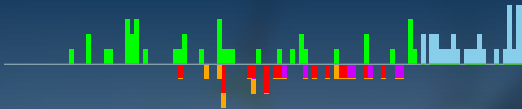


C: Canal Crossing

Michael Zündorf

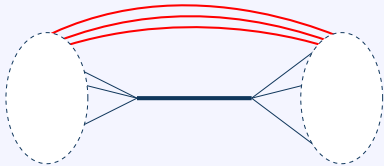
Problem

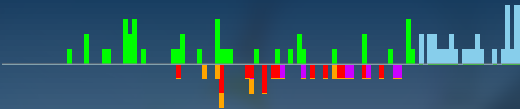
Given a tree and a set of extra edges (bridges), find the shortest tour through the extra edges, that uses each tree-edge at most once.



Solution

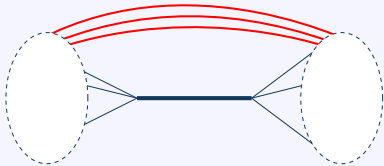
- Each tree edge connects two parts of the graph.

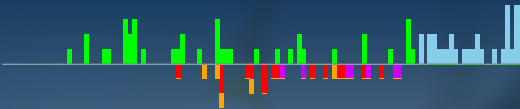




Solution

- Each tree edge connects two parts of the graph.
- If an odd number of bridges connect the two parts, the tree edge must be used in the optimal tour.



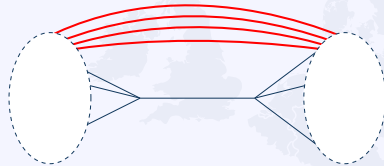
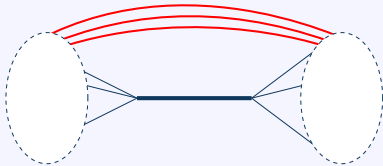


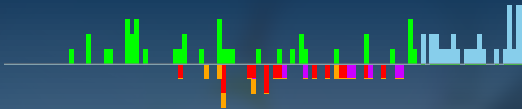
C: Canal Crossing

Michael Zündorf

Solution

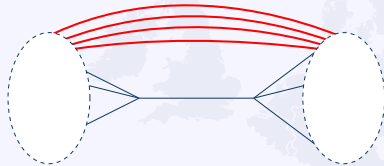
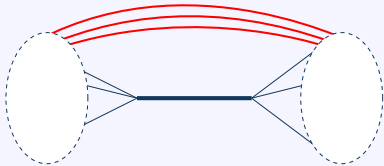
- Each tree edge connects two parts of the graph.
- If an odd number of bridges connect the two parts, the tree edge must be used in the optimal tour.
- The inverse is also true, if the number is even, the tree edge is not used in the shortest solution.

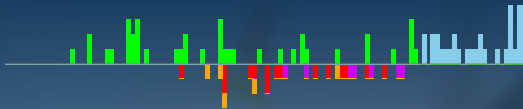




Solution

- Each tree edge connects two parts of the graph.
- If an odd number of bridges connect the two parts, the tree edge must be used in the optimal tour.
- The inverse is also true, if the number is even, the tree edge is not used in the shortest solution.
- Starting with leaves, count amount of bridges connected to that sub tree. Only the parity of this number matters.



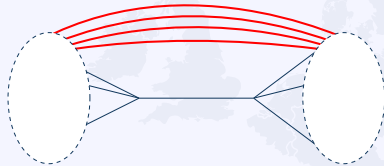
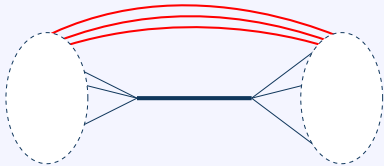


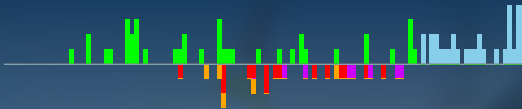
C: Canal Crossing

Michael Zündorf

Solution

- Each tree edge connects two parts of the graph.
- If an odd number of bridges connect the two parts, the tree edge must be used in the optimal tour.
- The inverse is also true, if the number is even, the tree edge is not used in the shortest solution.
- Starting with leaves, count amount of bridges connected to that sub tree. Only the parity of this number matters.
- This gives an $\mathcal{O}(n)$ DFS solution.



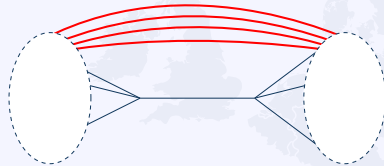
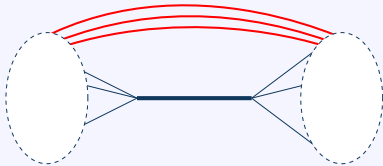


C: Canal Crossing

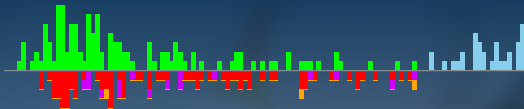
Michael Zündorf

Solution

- Each tree edge connects two parts of the graph.
- If an odd number of bridges connect the two parts, the tree edge must be used in the optimal tour.
- The inverse is also true, if the number is even, the tree edge is not used in the shortest solution.
- Starting with leaves, count amount of bridges connected to that sub tree. Only the parity of this number matters.
- This gives an $\mathcal{O}(n)$ DFS solution.

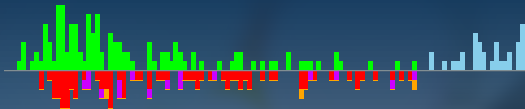


Statistics: 99 submissions, 39 accepted, 33 unknown



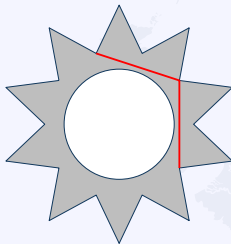
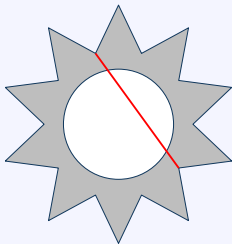
Problem

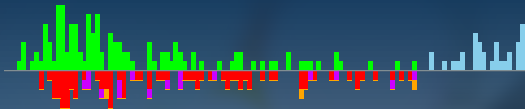
Wrap yarn around a wheel with n notches, by taking steps of k notches each time.
Given n , what value of k maximizes the amount of yarn used?



Observations

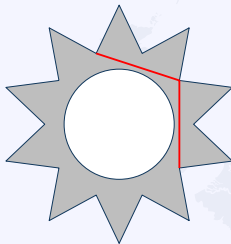
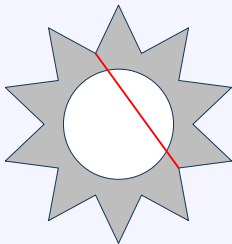
- If $\gcd(k, n) > 1$, then replacing k by $\frac{k}{\gcd(k, n)}$ increases the length.

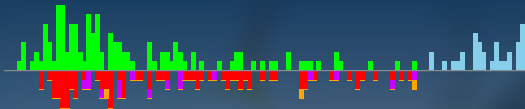




Observations

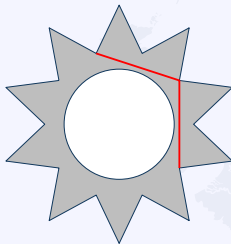
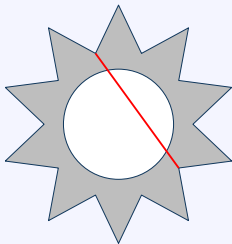
- If $\gcd(k, n) > 1$, then replacing k by $\frac{k}{\gcd(k, n)}$ increases the length.
- If $\gcd(k, n) = 1$, then the total length is $n \cdot (\text{length of single chord})$, which is maximized when k is as close to $\frac{n}{2}$ as possible.

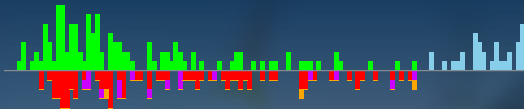




Observations

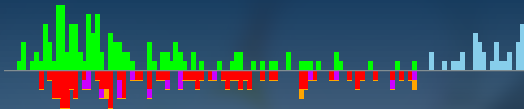
- If $\gcd(k, n) > 1$, then replacing k by $\frac{k}{\gcd(k, n)}$ increases the length.
- If $\gcd(k, n) = 1$, then the total length is $n \cdot (\text{length of single chord})$, which is maximized when k is as close to $\frac{n}{2}$ as possible.
- Need to find k coprime with n which is as close to $\frac{n}{2}$ as possible.





Observation

Need to find k coprime with n which is as close to $\frac{n}{2}$ as possible.

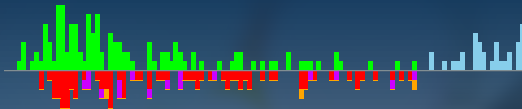


Observation

Need to find k coprime with n which is as close to $\frac{n}{2}$ as possible.

Solution

- Can use linear search, starting from $\frac{n}{2}$.

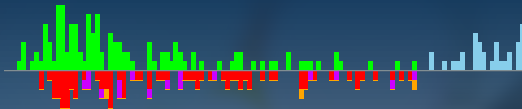


Observation

Need to find k coprime with n which is as close to $\frac{n}{2}$ as possible.

Solution

- Can use linear search, starting from $\frac{n}{2}$.
- Alternatively, use a direct formula:
 - if $n \equiv 1 \pmod 2$, take $k = \frac{n-1}{2}$,
 - if $n \equiv 2 \pmod 4$, take $k = \frac{n}{2} - 2$,
 - if $n \equiv 0 \pmod 4$, take $k = \frac{n}{2} - 1$.



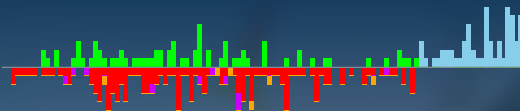
Observation

Need to find k coprime with n which is as close to $\frac{n}{2}$ as possible.

Solution

- Can use linear search, starting from $\frac{n}{2}$.
- Alternatively, use a direct formula:
 - if $n \equiv 1 \pmod{2}$, take $k = \frac{n-1}{2}$,
 - if $n \equiv 2 \pmod{4}$, take $k = \frac{n}{2} - 2$,
 - if $n \equiv 0 \pmod{4}$, take $k = \frac{n}{2} - 1$.

Statistics: 248 submissions, 119 accepted, 32 unknown

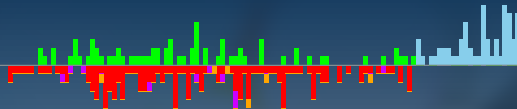


E: Erratic Lights

Atli Fannar Franklín

Problem

Every time you touch one of the $n \leq 100$ light bulbs, it randomly selects a new colour (red/green/blue), each with equal probability. What is the expected number of times you need to touch a light bulb to make all of them have the same colour?



E: Erratic Lights

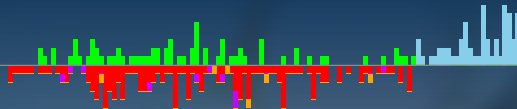
Atli Fannar Franklín

Problem

Every time you touch one of the $n \leq 100$ light bulbs, it randomly selects a new colour (red/green/blue), each with equal probability. What is the expected number of times you need to touch a light bulb to make all of them have the same colour?

The easy cases

- If all lights are equal, output 0.



E: Erratic Lights

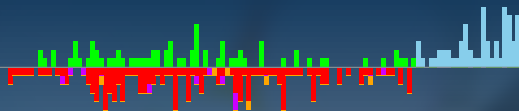
Atli Fannar Franklín

Problem

Every time you touch one of the $n \leq 100$ light bulbs, it randomly selects a new colour (red/green/blue), each with equal probability. What is the expected number of times you need to touch a light bulb to make all of them have the same colour?

The easy cases

- If all lights are equal, output 0.
- If there are only two different colours initially, change the least-occurring colours to the most-occurring ones. This takes expected $3x$ touches (where x is the initial count of the least-occurring colour).



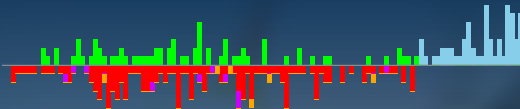
E: Erratic Lights

Atli Fannar Franklín

Solution for three different colours

Say the number of occurrences of each colour are $a \leq b \leq c$.

- Change the coloured light with the fewest occurrences to one of the other two. Disregarding which colour they change into, this takes expected $\frac{3}{2}a$ touches.



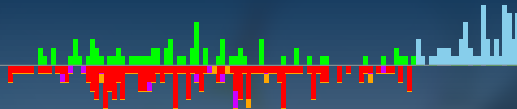
E: Erratic Lights

Atli Fannar Franklín

Solution for three different colours

Say the number of occurrences of each colour are $a \leq b \leq c$.

- Change the coloured light with the fewest occurrences to one of the other two. Disregarding which colour they change into, this takes expected $\frac{3}{2}a$ touches.
- After this, there are 2^a possible ways in which these first a lights turned into the other two colours of lights (exhaustive search is too slow, because a is up to 33).



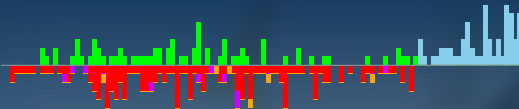
E: Erratic Lights

Atli Fannar Franklín

Solution for three different colours

Say the number of occurrences of each colour are $a \leq b \leq c$.

- Change the coloured light with the fewest occurrences to one of the other two. Disregarding which colour they change into, this takes expected $\frac{3}{2}a$ touches.
- After this, there are 2^a possible ways in which these first a lights turned into the other two colours of lights (exhaustive search is too slow, because a is up to 33).
- Only the *number* of lights in each colour is relevant, so we combine them. Say that i of the a lights turned into colour b , this happens $\binom{a}{i}$ times. Change the minimum of the two remaining colours to the final colour, this takes expected 3 touches each.

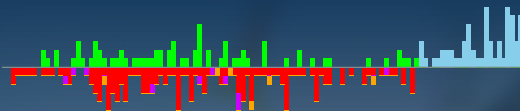


Solution for three different colours

Say the number of occurrences of each colour are $a \leq b \leq c$.

- Change the coloured light with the fewest occurrences to one of the other two. Disregarding which colour they change into, this takes expected $\frac{3}{2}a$ touches.
- After this, there are 2^a possible ways in which these first a lights turned into the other two colours of lights (exhaustive search is too slow, because a is up to 33).
- Only the *number* of lights in each colour is relevant, so we combine them. Say that i of the a lights turned into colour b , this happens $\binom{a}{i}$ times. Change the minimum of the two remaining colours to the final colour, this takes expected 3 touches each.
- The final expected number of touches is:

$$\frac{3}{2}a + \frac{1}{2^a} \sum_{i=0}^a \binom{a}{i} 3 \min(b+i, c+(a-i))$$



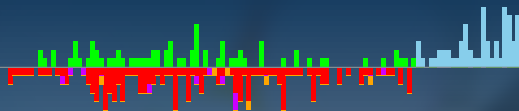
E: Erratic Lights

Atli Fannar Franklín

Alternative solutions

- Dynamic Programming, anything up to $\mathcal{O}(n^3)$ will pass.



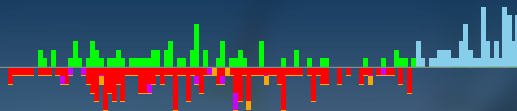


E: Erratic Lights

Atli Fannar Franklín

Alternative solutions

- Dynamic Programming, anything up to $\mathcal{O}(n^3)$ will pass.
- Simulate backwards for 1000-ish steps, and output the expected number of occurrences of (a, b, c) .



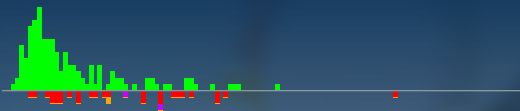
E: Erratic Lights

Atli Fannar Franklín

Alternative solutions

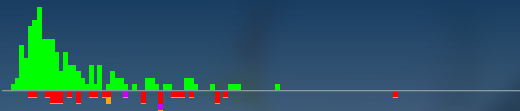
- Dynamic Programming, anything up to $\mathcal{O}(n^3)$ will pass.
- Simulate backwards for 1000-ish steps, and output the expected number of occurrences of (a, b, c) .

Statistics: 286 submissions, 72 accepted, 63 unknown



Problem

- There are n people who each have a_i cash and a bill of b_i .
- Choose a person i which does not pay more than b_i if they pay the bill and get all cash from others.

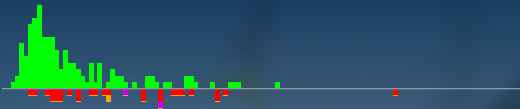


Problem

- There are n people who each have a_i cash and a bill of b_i .
- Choose a person i which does not pay more than b_i if they pay the bill and get all cash from others.

Observation

- The i th person can pay, if: $(\sum_{j=1}^n b_j) - (\sum_{j=1}^n a_j) + a_i \leq b_i$.
- This is equivalent to: $(\sum_{j=1}^n b_j) - (\sum_{j=1}^n a_j) \leq b_i - a_i$



Problem

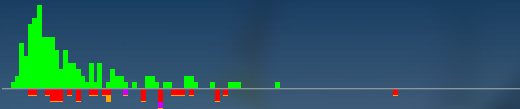
- There are n people who each have a_i cash and a bill of b_i .
- Choose a person i which does not pay more than b_i if they pay the bill and get all cash from others.

Observation

- The i th person can pay, if: $(\sum_{j=1}^n b_j) - (\sum_{j=1}^n a_j) + a_i \leq b_i$.
- This is equivalent to: $(\sum_{j=1}^n b_j) - (\sum_{j=1}^n a_j) \leq b_i - a_i$

Solution

- Precompute $S = (\sum_{j=1}^n b_j) - (\sum_{j=1}^n a_j)$.
- For each i check whether $S \leq b_i - a_i$.
- Runtime: $\mathcal{O}(n)$.



Problem

- There are n people who each have a_i cash and a bill of b_i .
- Choose a person i which does not pay more than b_i if they pay the bill and get all cash from others.

Observation

- The i th person can pay, if: $(\sum_{j=1}^n b_j) - (\sum_{j=1}^n a_j) + a_i \leq b_i$.
- This is equivalent to: $(\sum_{j=1}^n b_j) - (\sum_{j=1}^n a_j) \leq b_i - a_i$

Solution

- Precompute $S = (\sum_{j=1}^n b_j) - (\sum_{j=1}^n a_j)$.
- For each i check whether $S \leq b_i - a_i$.
- Runtime: $\mathcal{O}(n)$.

Problem

Given a permutation, reorder as few elements as possible to make it mountain-shaped (first increasing then decreasing) in $\mathcal{O}(n \log n)$.

Problem

Given a permutation, reorder as few elements as possible to make it mountain-shaped (first increasing then decreasing) in $\mathcal{O}(n \log n)$.

First insight

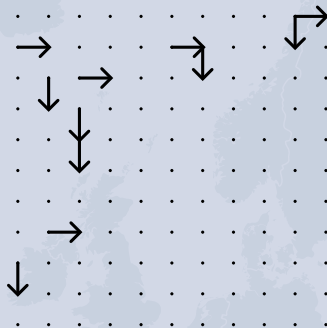
If we place elements from smallest to largest, each element will go to one of the borders. This leads to a quadratic DP with state space

(Num elements placed on the left so far, Num elements placed on the right so far).

Transition value is 0 if element is misplaced, and 1 if it stays where it is.

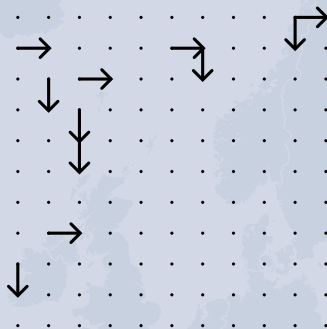
Need to find path from top left to bottom right that

- only moves down and right, and
- hits as many arrows as possible.



Need to find path from top left to bottom right that

- Key insight: there are at most $2n$ arrows!



Visualizing the DP state space

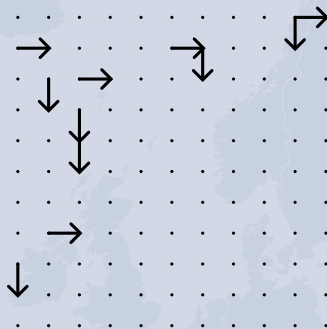
Need to find path from top left to bottom right that

- only moves down and right, and
- hits as many arrows as possible.

Key insight: there are at most $2n$ arrows!

Possible approaches:

- Rephrase DP as Longest Increasing Subsequence.
- Quickly simulate DP using segment tree or ordered set.



Visualizing the DP state space

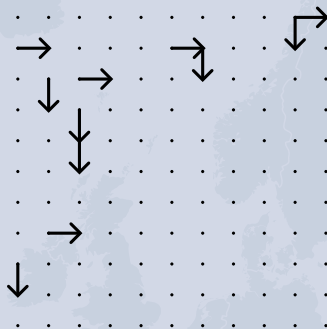
Need to find path from top left to bottom right that

- only moves down and right, and
- hits as many arrows as possible.

Key insight: there are at most $2n$ arrows!

Possible approaches:

- Rephrase DP as Longest Increasing Subsequence.
- Quickly simulate DP using segment tree or ordered set.



Problem

Find an implicit matching on the set of n bit binary strings with k ones, or report that no such matching exists. Two binary strings are adjacent if one can be transformed into the other by swapping the values at two distinct indices.

Find an implicit matching on the set of n bit binary strings with k ones, or report that no such matching exists. Two binary strings are adjacent if one can be transformed into the other by swapping the values at two distinct indices.



Construction

There are many solutions. One is to find a Hamiltonian path, then match the i -th value to the $(i \oplus 1)$ -th.

- Build the Hamiltonian path $H(n, k)$ recursively.
- $H(n, k) = 0H(n-1, k) + 1H^R(n-1, k-1)$ where H^R is the reversed Hamiltonian path.

0	00001111
	... $H(n-1, k)$

0	10000111
---	----------

1	10000011
	... $H^R(n-1, k-1)$

1	00000111
---	----------



Runtime

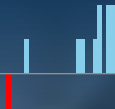
Runtime $\mathcal{O}(n)$ if done well, $\mathcal{O}(n^2)$ also passes.



Runtime

Runtime $\mathcal{O}(n)$ if done well, $\mathcal{O}(n^2)$ also passes.

Statistics: 37 submissions, 5 accepted, 28 unknown

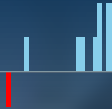


I: Illuminated Stalls

Jeroen Op de Beek

Problem

- Given $n \leq 2 \cdot 10^5$ axis-aligned lines in 2D.
- Horizontal lines do not touch horizontal ones, same for vertical.
- Move one line to form a square.



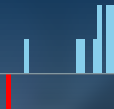
Problem

- Given $n \leq 2 \cdot 10^5$ axis-aligned lines in 2D.
- Horizontal lines do not touch horizontal ones, same for vertical.
- Move one line to form a square.

Solution

- If there exist a rectangle, it is always possible.
- If there are more than $2n$ intersections, there exists a rectangle.
- We only look for U-shapes, where the new line is moved to the top.
- For a horizontal line look at its intersections.
- Fix the smaller of the two vertical lines.
- We only need to check the closest taller vertical line to the left and right.





I: Illuminated Stalls

Jeroen Op de Beek

Implementation

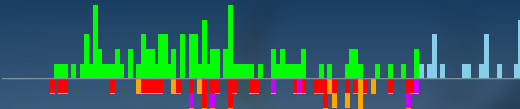
- For simplicity, try all four 90 degree rotations separately.
- Calculate all intersections via a swepline, quit if there are too many.
- Iterate over horizontal lines, sort intersecting vertical lines by decreasing height.
- Sweepline over the vertical line, lookup the neighboring lines in a set.
- Check if there already are upper horizontal lines at the correct height.
- Check which length are currently used and if the remaining longest one is enough.
- Careful: you may need to move one of the upper horizontal lines.



Implementation

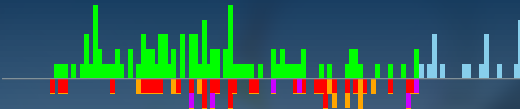
- For simplicity, try all four 90 degree rotations separately.
- Calculate all intersections via a sweepline, quit if there are too many.
- Iterate over horizontal lines, sort intersecting vertical lines by decreasing height.
- Sweepline over the vertical line, lookup the neighboring lines in a set.
- Check if there already are upper horizontal lines at the correct height.
- Check which length are currently used and if the remaining longest one is enough.
- Careful: you may need to move one of the upper horizontal lines.

Statistics: 11 submissions, 0 accepted, 10 unknown



Problem

Given n people sharing an apartment with k keys going on q trips, find who needs to take a key such that no one arrives to an empty apartment without a key.

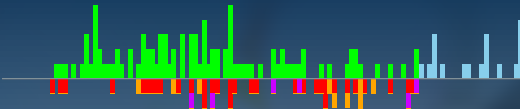


Problem

Given n people sharing an apartment with k keys going on q trips, find who needs to take a key such that no one arrives to an empty apartment without a key.

Observation

A trip only needs a key if it ends when the apartment is empty.

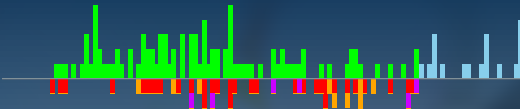


Problem

Given n people sharing an apartment with k keys going on q trips, find who needs to take a key such that no one arrives to an empty apartment without a key.

Solution

- Simulate while keeping track of the number of people in the apartment.
 - Loop through a sorted array of all arrivals and departures.
 - If a person arrives to an empty apartment, they should have taken a key.

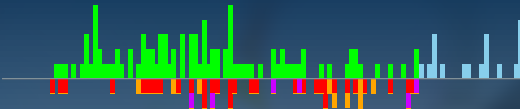


Problem

Given n people sharing an apartment with k keys going on q trips, find who needs to take a key such that no one arrives to an empty apartment without a key.

Solution

- Simulate while keeping track of the number of people in the apartment.
 - Loop through a sorted array of all arrivals and departures.
 - If a person arrives to an empty apartment, they should have taken a key.
- Do a second simulation while keeping track of the number of keys in the apartment.
 - If this ever becomes negative, output “impossible”.

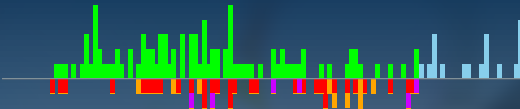


Problem

Given n people sharing an apartment with k keys going on q trips, find who needs to take a key such that no one arrives to an empty apartment without a key.

Solution

- Simulate while keeping track of the number of people in the apartment.
 - Loop through a sorted array of all arrivals and departures.
 - If a person arrives to an empty apartment, they should have taken a key.
- Do a second simulation while keeping track of the number of keys in the apartment.
 - If this ever becomes negative, output “impossible”.
- Runtime: $\mathcal{O}(n \log n)$.

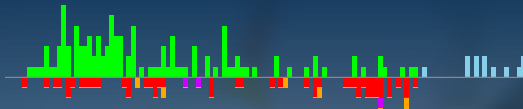


Problem

Given n people sharing an apartment with k keys going on q trips, find who needs to take a key such that no one arrives to an empty apartment without a key.

Solution

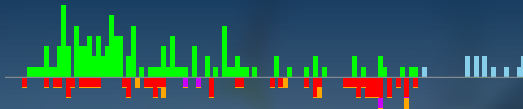
- Simulate while keeping track of the number of people in the apartment.
 - Loop through a sorted array of all arrivals and departures.
 - If a person arrives to an empty apartment, they should have taken a key.
- Do a second simulation while keeping track of the number of keys in the apartment.
 - If this ever becomes negative, output “impossible”.
- Runtime: $\mathcal{O}(n \log n)$.



Problem

Construct an $h \times w$ grid of the letters 'K', 'I' and 'T' such that each letter occurs a given number of times and the word "KIT" appears exactly once when reading in the 8 possible directions.

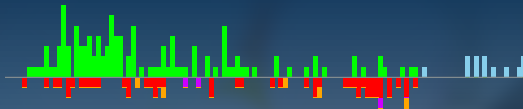
I	K	I	I	T
K	K	T	K	T
I	T	I	T	I
K	T	T	K	I



Solution

- Start with “KIT” in the top left corner.
- Then place all the remaining ‘T’s, followed by all the ‘K’s and finally all the ‘I’s.
- No extra occurrences of “KIT” possible, as no ‘I’ can be between a ‘K’ and a ‘T’.

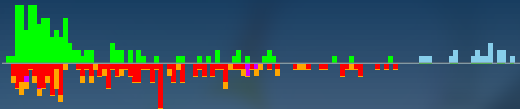
K	I	T	T	T
T	T	T	T	K
K	K	K	K	I
I	I	I	I	I



Solution

- Start with “KIT” in the top left corner.
- Then place all the remaining ‘T’s, followed by all the ‘K’s and finally all the ‘I’s.
- No extra occurrences of “KIT” possible, as no ‘I’ can be between a ‘K’ and a ‘T’.

K	I	T	T	T
T	T	T	T	K
K	K	K	K	I
I	I	I	I	I

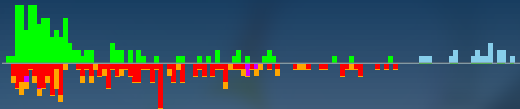


L: Last Christmas

Jeroen Bransen

Problem

Given n top-10 lists, which artist appears most often? If there is a tie, output who is most often number 1. If there is a tie, output who is most often number 2 and so on.



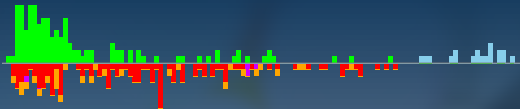
Problem

Given n top-10 lists, which artist appears most often? If there is a tie, output who is most often number 1. If there is a tie, output who is most often number 2 and so on.

Solution

- Count the appearances for each artist.
- Count for each artist and for each k how many number k hits they have.
- Find the best artist, do the tiebreaking according to the rules.

Runtime: $\mathcal{O}(n)$



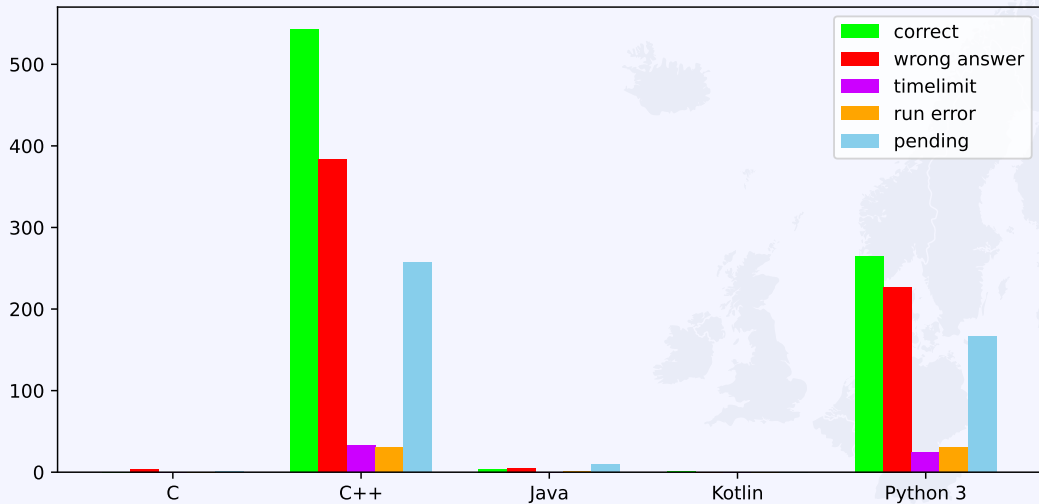
Problem

Given n top-10 lists, which artist appears most often? If there is a tie, output who is most often number 1. If there is a tie, output who is most often number 2 and so on.

Solution

- Count the appearances for each artist.
- Count for each artist and for each k how many number k hits they have.
- Find the best artist, do the tiebreaking according to the rules.

Runtime: $\mathcal{O}(n)$



Jury work

- 658 commits (including test session) (last year: 933)

Jury work

- 658 commits (including test session) (last year: 933)
- 1413 secret test cases (last year: 1281) ($\approx 117\frac{3}{4}$ per problem!)

Jury work

- 658 commits (including test session) (last year: 933)
- 1413 secret test cases (last year: 1281) ($\approx 117\frac{3}{4}$ per problem!)
- 317 jury/proofreader solutions (last year: 356)

Jury work

- 658 commits (including test session) (last year: 933)
- 1413 secret test cases (last year: 1281) ($\approx 117\frac{3}{4}$ per problem!)
- 317 jury/proofreader solutions (last year: 356)
- The minimum¹ number of lines the jury needed to solve all problems is:

$$8 + 7 + 14 + 1 + 2 + 2 + 8 + 22 + 45 + 5 + 3 + 5 = 122$$

On average $10\frac{1}{6}$ lines per problem, down from 18 last year.

¹With some code golfing, last year we golfed less