Southern and Volga Russian Regional Contest

Saratov State University

18 ноября 2024 г.

J. Waiting for...

Events of two types occur: a bus arrives with b free seats and p people come to the stop. It is required to determine whether there will be free seats on each bus.

J. Waiting for...

- We will maintain the current number of people at the stop s

J. Waiting for...

- We will maintain the current number of people at the stop s
- If p people arrive, we will do $s \mathrel{+}= p$

J. Waiting for...

- We will maintain the current number of people at the stop s
- If p people arrive, we will do $s += p$
- If a bus arrives with b seats, we will compare b and s

J. Waiting for...

- We will maintain the current number of people at the stop s
- If p people arrive, we will do $s \mathrel{+}= p$
- If a bus arrives with b seats, we will compare b and s
- If $b > s$, the answer is `YES`, otherwise `NO`

J. Waiting for...

- We will maintain the current number of people at the stop s
- If p people arrive, we will do $s += p$
- If a bus arrives with b seats, we will compare b and s
- If $b > s$, the answer is `YES`, otherwise `NO`
- Then we will do $s -= \min(b, s)$

N. Fixing the Expression

There is a logical expression of the form “one digit is less/greater/equal to another”, change the minimum number of symbols in it to make it true.

N. Fixing the Expression

- You can change the sign between the digits instead of changing the digits

N. Fixing the Expression

- You can change the sign between the digits instead of changing the digits
- If the expression is already correct, the sign will change to the same one, meaning there will be no changes

L. Bridge Renovation

Break planks of length 60 into 18, 21, and 25, n planks of each length.

- Greedy ideas

L. Bridge Renovation

Break planks of length 60 into 18, 21, and 25, n planks of each length.

- Greedy ideas
- Creating pairs $25 + 25$ is profitable

L. Bridge Renovation

Break planks of length 60 into 18, 21, and 25, n planks of each length.

- Greedy ideas
- Creating pairs $25 + 25$ is profitable
- Need $\left\lfloor \frac{n}{2} \right\rfloor$ pairs of $25 + 25$

L. Bridge Renovation

Break planks of length 60 into 18, 21, and 25, n planks of each length.

- Greedy ideas
- Creating pairs $25 + 25$ is profitable
- Need $\left\lfloor \frac{n}{2} \right\rfloor$ pairs of $25 + 25$
- Creating pairs $21 + 21 + 18$ is profitable

L. Bridge Renovation

Break planks of length 60 into 18, 21, and 25, n planks of each length.

- Greedy ideas
- Creating pairs $25 + 25$ is profitable
- Need $\left\lfloor \frac{n}{2} \right\rfloor$ pairs of $25 + 25$
- Creating pairs $21 + 21 + 18$ is profitable
- Need $\left\lfloor \frac{n}{2} \right\rfloor$ pairs of $21 + 21 + 18$

L. Bridge Renovation

Break planks of length 60 into 18, 21, and 25, n planks of each length.

- Greedy ideas
- Creating pairs $25 + 25$ is profitable
- Need $\left\lfloor \frac{n}{2} \right\rfloor$ pairs of $25 + 25$
- Creating pairs $21 + 21 + 18$ is profitable
- Need $\left\lfloor \frac{n}{2} \right\rfloor$ pairs of $21 + 21 + 18$
- Additionally, need (n mod 2) pairs of $25 + 21$

L. Bridge Renovation

Break planks of length 60 into 18, 21, and 25, n planks of each length.

- Greedy ideas
- Creating pairs $25 + 25$ is profitable
- Need $\lfloor \frac{n}{2} \rfloor$ pairs of $25 + 25$
- Creating pairs $21 + 21 + 18$ is profitable
- Need $\lfloor \frac{n}{2} \rfloor$ pairs of $21 + 21 + 18$
- Additionally, need $(n \bmod 2)$ pairs of $25 + 21$
- **For boards 25 and 21, exactly n planks of 60 are needed**

L. Bridge Renovation

Break planks of length 60 into 18, 21, and 25, n planks of each length.

- There are $\lceil \frac{n}{2} \rceil$ planks of 18 left

L. Bridge Renovation

Break planks of length 60 into 18, 21, and 25, n planks of each length.

- There are $\left\lceil \frac{n}{2} \right\rceil$ planks of 18 left
- Creating pairs $18 + 18 + 18$ is profitable

L. Bridge Renovation

Break planks of length 60 into 18, 21, and 25, n planks of each length.

- There are $\lceil \frac{n}{2} \rceil$ planks of 18 left
- Creating pairs $18 + 18 + 18$ is profitable
- Need $\left\lceil \frac{\lceil \frac{n}{2} \rceil}{3} \right\rceil = \lceil \frac{n}{6} \rceil$ pairs of $18 + 18 + 18$

L. Bridge Renovation

Break planks of length 60 into 18, 21, and 25, n planks of each length.

- There are $\left\lceil \frac{n}{2} \right\rceil$ planks of 18 left
- Creating pairs $18 + 18 + 18$ is profitable
- Need $\left\lceil \frac{\left\lceil \frac{n}{2} \right\rceil}{3} \right\rceil = \left\lceil \frac{n}{6} \right\rceil$ pairs of $18 + 18 + 18$

Answer to the problem: $n + \left\lceil \frac{n}{6} \right\rceil$

C. DIY

There is a list of numbers. You need to select 8 of them and use them as coordinates for the corners of a rectangle with sides parallel to the coordinate axes. The rectangle should have the maximum possible area.

C. DIY

- The numbers must definitely be taken in pairs because there must be 2 points on each horizontal/vertical line

C. DIY

- The numbers must definitely be taken in pairs because there must be 2 points on each horizontal/vertical line
- If there are no 4 pairs of identical elements, the answer is NO

C. DIY

- The numbers must definitely be taken in pairs because there must be 2 points on each horizontal/vertical line
- If there are no 4 pairs of identical elements, the answer is NO
- Otherwise, let's "compress" the list so that if there were x occurrences in the original, it will become $\lfloor \frac{x}{2} \rfloor$ occurrences in the compressed list

C. DIY

- Sort the compressed list. Let's call the new list b, and its length m

C. DIY

- Sort the compressed list. Let's call the new list b, and its length m
- On each axis, the smaller the "left" coordinate and the larger the "right", the larger the size of the rectangle

C. DIY

- Sort the compressed list. Let's call the new list b, and its length m
- On each axis, the smaller the "left" coordinate and the larger the "right", the larger the size of the rectangle
- You can always take elements b_1, b_2, b_{m-1}, b_m

C. DIY

- Sort the compressed list. Let's call the new list b, and its length m
- On each axis, the smaller the “left” coordinate and the larger the “right”, the larger the size of the rectangle
- You can always take elements b_1, b_2, b_{m-1}, b_m
- It can be proven that it is profitable to take pairs (b_1, b_{m-1}) and (b_2, b_m)

A. Bonus Project

n developers are working on the project. To complete it, they must spend a total of k hours. The developers take turns naming an integer number of hours c_i that each will work on the project. If they complete the project, the i-th will receive $a_i - c_i \cdot b_i$ burles. How many hours will each name if each developer wants to maximize their profit?

A. Bonus Project

- Each developer has a limit on the number of hours, after which they work at a loss

A. Bonus Project

- Each developer has a limit on the number of hours, after which they work at a loss
- If the sum of the limits is less than k, the answer for all is 0

A. Bonus Project

- Each developer has a limit on the number of hours, after which they work at a loss
- If the sum of the limits is less than k, the answer for all is 0
- Let's solve the problem from the last developer to the first

A. Bonus Project

- Each developer has a limit on the number of hours, after which they work at a loss
- If the sum of the limits is less than k, the answer for all is 0
- Let's solve the problem from the last developer to the first
- The last developer knows how many hours are left to work on the project

A. Bonus Project

- Each developer has a limit on the number of hours, after which they work at a loss
- If the sum of the limits is less than k, the answer for all is 0
- Let's solve the problem from the last developer to the first
- The last developer knows how many hours are left to work on the project
- If their limit is less, they will not work, and no one will receive profit

A. Bonus Project

- It is in the interest of the second-to-last developer to work as little as possible, but enough so that the last developer does not refuse to work

A. Bonus Project

- It is in the interest of the second-to-last developer to work as little as possible, but enough so that the last developer does not refuse to work
- It is optimal for the second-to-last to leave an amount of work exactly equal to the limit of the last developer

A. Bonus Project

- It is in the interest of the second-to-last developer to work as little as possible, but enough so that the last developer does not refuse to work
- It is optimal for the second-to-last to leave an amount of work exactly equal to the limit of the last developer
- The last developer will work $\min(k, \frac{a_i}{b_i})$

A. Bonus Project

- It is in the interest of the second-to-last developer to work as little as possible, but enough so that the last developer does not refuse to work
- It is optimal for the second-to-last to leave an amount of work exactly equal to the limit of the last developer
- The last developer will work $\min(k, \frac{a_i}{b_i})$
- Remove the last developer, subtract their hours from k, and move to $n - 1$

A. Bonus Project

- It is in the interest of the second-to-last developer to work as little as possible, but enough so that the last developer does not refuse to work
- It is optimal for the second-to-last to leave an amount of work exactly equal to the limit of the last developer
- The last developer will work $\min(k, \frac{a_i}{b_i})$
- Remove the last developer, subtract their hours from k, and move to $n - 1$

Time complexity: $O(n)$

G. Guess One Character

The jury has a binary string, and you need to guess one character in it. You can ask no more than 3 queries of the form “how many times some string occurs in the jury string as a substring?”

G. Guess One Character

- For each character, except the last, there is a next character — 1 or 0

G. Guess One Character

- For each character, except the last, there is a next character — 1 or 0
- Based on this, we can ask the following queries — 1, 11, and 10

G. Guess One Character

- For each character, except the last, there is a next character — 1 or 0
- Based on this, we can ask the following queries — 1, 11, and 10
- After each 1, except the last, there will be a character, so if the answer to the first query equals the sum of the other two, then the last character is not 1

B. Make It Equal

There is an integer array. Make the array consist of identical non-negative numbers, using the minimum number of operations of the type “decrease an element by 2 and increase the next (cyclically) by 1”.

B. Make It Equal

- If each element is equal to k, then with an additional n operations you can also get the value $(k-1)$

B. Make It Equal

- If each element is equal to k, then with an additional n operations you can also get the value $(k - 1)$
- You can use binary search on the final value of the array

B. Make It Equal

- To check if a value is reachable in binary search, we will cyclically fix (apply operations to make the element no more than k) elements until the sum of the array becomes less than or equal to $k \cdot n$

B. Make It Equal

- To check if a value is reachable in binary search, we will cyclically fix (apply operations to make the element no more than k) elements until the sum of the array becomes less than or equal to $k \cdot n$
- With such fixes, we perform operations that definitely need to be done

B. Make It Equal

- To check if a value is reachable in binary search, we will cyclically fix (apply operations to make the element no more than k) elements until the sum of the array becomes less than or equal to $k \cdot n$
- With such fixes, we perform operations that definitely need to be done
- Such a series of fixes converges in $O(n + \log A)$, because after the first round of fixes we will have only one element greater than the required value

K. Grid Walk

Find the cheapest path from $(1, 1)$ to (n, n), where $c(i, j) = \gcd(i, a) + \gcd(j, b)$.

- $P(x, y)$ is some path from $(1, 1)$ to (x, y)

K. Grid Walk

Find the cheapest path from $(1,1)$ to (n,n), where $c(i,j) = \gcd(i,a) + \gcd(j,b)$.

- $P(x,y)$ is some path from $(1,1)$ to (x,y)
- $\sum\limits_{(i,j)\in P} c(i,j) = \sum\limits_{(i,j)\in P} \gcd(i,a) + \gcd(j,b)$

K. Grid Walk

Find the cheapest path from $(1, 1)$ to (n, n), where
$c(i, j) = \gcd(i, a) + \gcd(j, b)$.

- $P(x, y)$ is some path from $(1, 1)$ to (x, y)
- $\sum\limits_{(i,j)\in P} c(i, j) = \sum\limits_{(i,j)\in P} \gcd(i, a) + \gcd(j, b)$
- $\sum\limits_{(i,j)\in P} c(i, j) \geq (x - 1) + (y - 1) + \sum_{i=1}^{x} \gcd(i, a) + \sum\limits_{j=1}^{y} \gcd(j, b)$

K. Grid Walk

Find the cheapest path from $(1,1)$ to (n,n), where $c(i,j) = \gcd(i,a) + \gcd(j,b)$.

- $P(x,y)$ is some path from $(1,1)$ to (x,y)
- $\sum\limits_{(i,j)\in P} c(i,j) = \sum\limits_{(i,j)\in P} \gcd(i,a) + \gcd(j,b)$
- $\sum\limits_{(i,j)\in P} c(i,j) \geq (x-1) + (y-1) + \sum_{i=1}^{x} \gcd(i,a) + \sum\limits_{j=1}^{y} \gcd(j,b)$
- If $\gcd(x,a) = 1$ or $\gcd(y,b) = 1$, then the lower bound **is achieved**

K. Grid Walk

- Let x be the maximum integer such that $x \leq n$ and $\gcd(x, a) = 1$

K. Grid Walk

- Let x be the maximum integer such that $x \le n$ and $\gcd(x, a) = 1$
- Let y be the maximum integer such that $y \le n$ and $\gcd(y, b) = 1$

K. Grid Walk

- Let x be the maximum integer such that $x \leq n$ and $\gcd(x, a) = 1$
- Let y be the maximum integer such that $y \leq n$ and $\gcd(y, b) = 1$
- There exists an optimal path passing through (x, y)

K. Grid Walk

- Let x be the maximum integer such that $x \le n$ and $\gcd(x, a) = 1$
- Let y be the maximum integer such that $y \le n$ and $\gcd(y, b) = 1$
- There exists an optimal path passing through (x, y)
- The cost of the path $(1, 1) - (x, y)$ is already known

K. Grid Walk

- Let x be the maximum integer such that $x \leq n$ and $\gcd(x, a) = 1$
- Let y be the maximum integer such that $y \leq n$ and $\gcd(y, b) = 1$
- There exists an optimal path passing through (x, y)
- The cost of the path $(1, 1) - (x, y)$ is already known
- The cost of the path $(x, y) - (n, n)$ needs to be found

K. Grid Walk

- Let x be the maximum integer such that $x \le n$ and $\gcd(x, a) = 1$
- Let y be the maximum integer such that $y \le n$ and $\gcd(y, b) = 1$
- There exists an optimal path passing through (x, y)
- The cost of the path $(1, 1) - (x, y)$ is already known
- The cost of the path $(x, y) - (n, n)$ needs to be found
- For $n \le 10^6$, $n - x \le 25$ and $n - y \le 25$

K. Grid Walk

- Let x be the maximum integer such that $x \le n$ and $\gcd(x, a) = 1$
- Let y be the maximum integer such that $y \le n$ and $\gcd(y, b) = 1$
- There exists an optimal path passing through (x, y)
- The cost of the path $(1, 1) - (x, y)$ is already known
- The cost of the path $(x, y) - (n, n)$ needs to be found
- For $n \le 10^6$, $n - x \le 25$ and $n - y \le 25$
- $dp[26][26]$

K. Grid Walk

- Let x be the maximum integer such that $x \leq n$ and $\gcd(x, a) = 1$
- Let y be the maximum integer such that $y \leq n$ and $\gcd(y, b) = 1$
- There exists an optimal path passing through (x, y)
- The cost of the path $(1, 1) - (x, y)$ is already known
- The cost of the path $(x, y) - (n, n)$ needs to be found
- For $n \leq 10^6$, $n - x \leq 25$ and $n - y \leq 25$
- $dp[26][26]$

Time complexity: $O(n \log n)$

I. Polyathlon

A competition of m sports is taking place, in which n athletes participate. A binary matrix is given — is the i-th participant strong in the j-th sport? Sports $x, x+1, \ldots$ are played **cyclically**, after each sport, weak participants are eliminated if there is at least one strong participant. For each starting x, output the winner.

I. Polyathlon

- Consider it as a string problem

I. Polyathlon

- Consider it as a string problem
- The athlete who is lexicographically maximum wins

I. Polyathlon

- Consider it as a string problem
- The athlete who is lexicographically maximum wins
- For each cyclic shift, find the maximum

I. Polyathlon

- We will maintain the entire decreasing order of rows

I. Polyathlon

- We will maintain the entire decreasing order of rows
- Having the order for $x+1$, we will recalculate for x

I. Polyathlon

- We will maintain the entire decreasing order of rows
- Having the order for $x + 1$, we will recalculate for x
- Athletes with 1 in position x move to the front in the same order
- Athletes with 0 in position x remain at the back in the same order

I. Polyathlon

- You can use radix sort

- You can use radix sort
- Each iteration takes $O(n)$

I. Polyathlon

- You can use radix sort
- Each iteration takes $O(n)$
- We will calculate the answer for $x = m$, performing m iterations

I. Polyathlon

- You can use radix sort
- Each iteration takes $O(n)$
- We will calculate the answer for $x = m$, performing m iterations
- We will perform another m iterations, keeping the maximum after each

I. Polyathlon

- You can use radix sort
- Each iteration takes $O(n)$
- We will calculate the answer for $x = m$, performing m iterations
- We will perform another m iterations, keeping the maximum after each

Time complexity: $O(nm)$

D. Divide OR Conquer

Given an array of n integers, you have to count the number of ways to cut it into segments such that the bitwise OR on the segments forms a non-decreasing sequence.

D. Divide OR Conquer

- Consider segments with the same right boundary

D. Divide OR Conquer

- Consider segments with the same right boundary
- There are $\log_2 10^9$ different ORs among such segments

D. Divide OR Conquer

- Consider segments with the same right boundary
- There are $\log_2 10^9$ different ORs among such segments
- Use dynamic programming $dp[i][x]$

D. Divide OR Conquer

- Consider segments with the same right boundary
- There are $\log_2 10^9$ different ORs among such segments
- Use dynamic programming $dp[i][x]$
- “How many ways to cut a prefix of length i into segments so that the OR of the last segment is exactly x?”

D. Divide OR Conquer

- For each l, find all pairs (r, x) — the minimum r for which the OR on $[l, r)$ equals x

D. Divide OR Conquer

- For each l, find all pairs (r, x) — the minimum r for which the OR on $[l, r)$ equals x
- Pairs for l can be obtained from pairs for $l + 1$

D. Divide OR Conquer

- For each l, find all pairs (r, x) — the minimum r for which the OR on $[l, r)$ equals x
- Pairs for l can be obtained from pairs for $l + 1$
- Sort pairs (r, x) in descending order of x

D. Divide OR Conquer

- We will calculate our dynamic programming forwards

D. Divide OR Conquer

- We will calculate our dynamic programming forwards
- Consider two adjacent pairs (r_j, x_j) and (r_{j+1}, x_{j+1})

D. Divide OR Conquer

- We will calculate our dynamic programming forwards
- Consider two adjacent pairs (r_j, x_j) and (r_{j+1}, x_{j+1})
- We need to add $\sum dp[i][x]$ for all $x \leq x_j$ to the dp values on the segment from r_j to r_{j+1}

D. Divide OR Conquer

- We will calculate our dynamic programming forwards
- Consider two adjacent pairs (r_j, x_j) and (r_{j+1}, x_{j+1})
- We need to add $\sum dp[i][x]$ for all $x \leq x_j$ to the dp values on the segment from r_j to r_{j+1}
- We will do delayed addition with sweep line technique

D. Divide OR Conquer

- We will calculate our dynamic programming forwards
- Consider two adjacent pairs (r_j, x_j) and (r_{j+1}, x_{j+1})
- We need to add $\sum dp[i][x]$ for all $x \leq x_j$ to the dp values on the segment from r_j to r_{j+1}
- We will do delayed addition with sweep line technique
- At position r_j, we will add, at position r_{j+1} we will subtract the same value

D. Divide OR Conquer

- At each position, we will maintain a list of the form (OR value, sum of dp values)

D. Divide OR Conquer

- At each position, we will maintain a list of the form (OR value, sum of dp values)
- We will update the list with events from the sweep line

D. Divide OR Conquer

- At each position, we will maintain a list of the form (OR value, sum of dp values)
- We will update the list with events from the sweep line
- With two pointers, we will calculate the values of all dynamics for the position

D. Divide OR Conquer

- At each position, we will maintain a list of the form (OR value, sum of dp values)
- We will update the list with events from the sweep line
- With two pointers, we will calculate the values of all dynamics for the position
- Solution complexity: $O(n \log n \log \log n)$

M. Royal Flush

There is a deck of cards from 1 to 4 suits. We take the top 5 cards from it and on each turn can choose which cards to discard and which to keep. After each discard, we take the next cards until we have 5 cards in hand. The task is to obtain a royal flush in the minimum expected number of moves.

M. Royal Flush

- There are $n+1$ types of cards: cards of each of the n suits involved in constructing the royal flush, and all others

M. Royal Flush

- There are $n+1$ types of cards: cards of each of the n suits involved in constructing the royal flush, and all others
- Cards of types not required in the royal flush can be discarded immediately, but for other types, it is more complicated

M. Royal Flush

- There are $n + 1$ types of cards: cards of each of the n suits involved in constructing the royal flush, and all others
- Cards of types not required in the royal flush can be discarded immediately, but for other types, it is more complicated
- For each state, we need to optimally choose which cards to discard — let's try to choose it with dynamic programming

M. Royal Flush

- The state can be described by the following parameters: how many cards are in the deck, how many cards of each type we have in hand

M. Royal Flush

- The state can be described by the following parameters: how many cards are in the deck, how many cards of each type we have in hand
- If we discard a card required for the royal flush, then the royal flush for that suit can no longer be collected; for each suit, we need to keep track of whether it can still be collected

M. Royal Flush

- The state can be described by the following parameters: how many cards are in the deck, how many cards of each type we have in hand
- If we discard a card required for the royal flush, then the royal flush for that suit can no longer be collected; for each suit, we need to keep track of whether it can still be collected
- For example, we can store -1 for the number of cards of a type that can no longer be collected, but for this, we will need to maintain the total number of cards in hand as an additional state

M. Royal Flush

- Transitions: if the number of cards in hand is less than 5, we take a random card (the dp value is equal to the weighted sum of dp values from the states we transition to)

M. Royal Flush

- Transitions: if the number of cards in hand is less than 5, we take a random card (the dp value is equal to the weighted sum of dp values from the states we transition to)
- If the number of cards in hand is 5, we will iterate over which cards to discard (the dp value is equal to $1+$ the minimum of the dp values we transition to)

M. Royal Flush

- Transitions: if the number of cards in hand is less than 5, we take a random card (the dp value is equal to the weighted sum of dp values from the states we transition to)
- If the number of cards in hand is 5, we will iterate over which cards to discard (the dp value is equal to $1+$ the minimum of the dp values we transition to)
- If it works too slowly, we can precalculate the answers locally

There are n barrels. We throw clay at the bottom of the barrels to maximize the volume of water in the first barrel.

- Greedy ideas

E. Barrels

There are n barrels. We throw clay at the bottom of the barrels to maximize the volume of water in the first barrel.

- Greedy ideas
- It is profitable to fill the barrels from right to left.

E. Barrels

There are n barrels. We throw clay at the bottom of the barrels to maximize the volume of water in the first barrel.

- Greedy ideas
- It is profitable to fill the barrels from right to left.
- Simplifying the procedure:

E. Barrels

There are n barrels. We throw clay at the bottom of the barrels to maximize the volume of water in the first barrel.

- Greedy ideas
- It is profitable to fill the barrels from right to left.
- Simplifying the procedure:
  - throw $h[i-1]$ units of clay immediately

E. Barrels

There are n barrels. We throw clay at the bottom of the barrels to maximize the volume of water in the first barrel.

- Greedy ideas
- It is profitable to fill the barrels from right to left.
- Simplifying the procedure:
  - throw $h[i-1]$ units of clay immediately
  - recalculate the balance

E. Barrels

There are n barrels. We throw clay at the bottom of the barrels to maximize the volume of water in the first barrel.

- Greedy ideas
- It is profitable to fill the barrels from right to left.
- Simplifying the procedure:
  - throw $h[i-1]$ units of clay immediately
  - recalculate the balance
  - seal the barrel with one additional unit of clay

E. Barrels

How to conveniently recalculate?

E. Barrels

How to conveniently recalculate?

- Maintain segments of communicating vessels and work with segments

E. Barrels

How to conveniently recalculate?

- Maintain segments of communicating vessels and work with segments
- In the general case, on the suffix, we store a “stack” of filled segments

E. Barrels

How to conveniently recalculate?

- Maintain segments of communicating vessels and work with segments
- In the general case, on the suffix, we store a “stack” of filled segments
- Water is collected in the last unfilled segment.

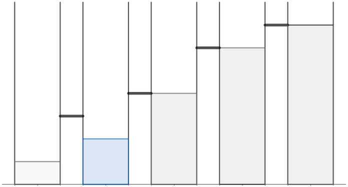
E. Barrels

How to conveniently recalculate?

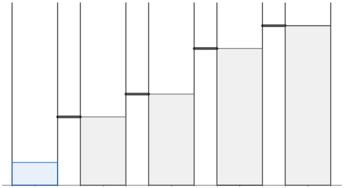
- Maintain segments of communicating vessels and work with segments
- In the general case, on the suffix, we store a “stack” of filled segments
- Water is collected in the last unfilled segment.
- $2 + 1$ cases:

E. Barrels

First case:

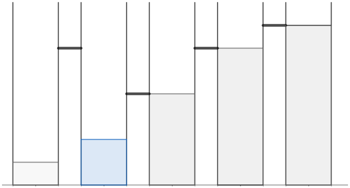


“The stack” expands

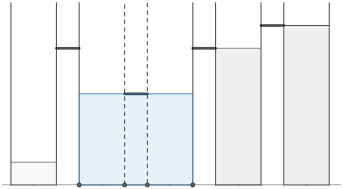


E. Barrels

Second case:



The segments merge



E. Barrels

- We store the “stack” in a *list*, we can add/remove from the beginning and remove from the end in $O(1)$

E. Barrels

- We store the "stack" in a *list*, we can add/remove from the beginning and remove from the end in $O(1)$
- Complexity: $O(n)$ in total.

F. Alternative Platforms

For each k, calculate the maximum of the minimums over all subsets of size k.

F. Alternative Platforms

For each k, calculate the maximum of the minimums over all subsets of size k.

Part one: for one value of k.

F. Alternative Platforms

For each k, calculate the maximum of the minimums over all subsets of size k.

Part one: for one value of k.

- Let $F(k) = \sum_S E(S)$, where $|S| = k$

For each k, calculate the maximum of the minimums over all subsets of size k.

Part one: for one value of k.

- Let $F(k) = \sum_S E(S)$, where $|S| = k$
- $$avg_k = \frac{F(k)}{\binom{n}{k}}$$

F. Alternative Platforms

For each k, calculate the maximum of the minimums over all subsets of size k.

Part one: for one value of k.

- Let $F(k) = \sum_S E(S)$, where $|S| = k$
- $$avg_k = \frac{F(k)}{\binom{n}{k}}$$
- $F(k) = \sum_{e=1}^{10^6} e \cdot |\{S \mid E(S) = e\}|$

F. Alternative Platforms

For each k, calculate the maximum of the minimums over all subsets of size k.

Part one: for one value of k.

- Let $F(k) = \sum_S E(S)$, where $|S| = k$
- $$avg_k = \frac{F(k)}{\binom{n}{k}}$$
- $F(k) = \sum_{e=1}^{10^6} e \cdot |\{S \mid E(S) = e\}|$
- $F(k) = \sum_{e=1}^{10^6} |\{S \mid E(S) \geq e\}|$

Calculating $|\{S \mid E(S) \geq e\}|$ using the inclusion/exclusion formula

F. Alternative Platforms

Calculating $|\{S \mid E(S) \geq e\}|$ using the inclusion/exclusion formula

- add $|\{S \mid \min_{s \in S} v(S) \geq e\}|$

F. Alternative Platforms

Calculating $|\{S \mid E(S) \geq e\}|$ using the inclusion/exclusion formula

- add $|\{S \mid \min_{s \in S} v(S) \geq e\}|$
- $+\binom{p_1}{k}$ for some p_1

F. Alternative Platforms

Calculating $|\{S \mid E(S) \geq e\}|$ using the inclusion/exclusion formula

- add $|\{S \mid \min_{s \in S} v(S) \geq e\}|$
- $+\binom{p_1}{k}$ for some p_1
- add $|\{S \mid \min_{s \in S} r(S) \geq e\}|$

F. Alternative Platforms

Calculating $|\{S \mid E(S) \geq e\}|$ using the inclusion/exclusion formula

- add $|\{S \mid \min_{s \in S} v(S) \geq e\}|$
- $+\binom{p_1}{k}$ for some p_1
- add $|\{S \mid \min_{s \in S} r(S) \geq e\}|$
- $+\binom{p_2}{k}$ for some p_2

F. Alternative Platforms

Calculating $|\{S \mid E(S) \geq e\}|$ using the inclusion/exclusion formula

- add $|\{S \mid \min_{s \in S} v(S) \geq e\}|$
- $+\binom{p_1}{k}$ for some p_1
- add $|\{S \mid \min_{s \in S} r(S) \geq e\}|$
- $+\binom{p_2}{k}$ for some p_2
- subtract $|\{S \mid \min_{s \in S} v(S) \text{ and } \min_{s \in S} v(S) \geq e\}|$

F. Alternative Platforms

Calculating $|\{S \mid E(S) \geq e\}|$ using the inclusion/exclusion formula

- add $|\{S \mid \min_{s \in S} v(S) \geq e\}|$
- $+\binom{p_1}{k}$ for some p_1
- add $|\{S \mid \min_{s \in S} r(S) \geq e\}|$
- $+\binom{p_2}{k}$ for some p_2
- subtract $|\{S \mid \min_{s \in S} v(S) \text{ and } \min_{s \in S} v(S) \geq e\}|$
- $m_i = \min(v_i, r_i)$

F. Alternative Platforms

Calculating $|\{S \mid E(S) \geq e\}|$ using the inclusion/exclusion formula

- add $|\{S \mid \min_{s \in S} v(S) \geq e\}|$
- $+\binom{p_1}{k}$ for some p_1
- add $|\{S \mid \min_{s \in S} r(S) \geq e\}|$
- $+\binom{p_2}{k}$ for some p_2
- subtract $|\{S \mid \min_{s \in S} v(S) \text{ and } \min_{s \in S} v(S) \geq e\}|$
- $m_i = \min(v_i, r_i)$
- $-\binom{p_3}{k}$ for some p_3

F. Alternative Platforms

$$F(k) = \sum_{i=1}^{n} f_i \binom{i}{k}$$

F. Alternative Platforms

$$F(k) = \sum_{i=1}^{n} f_i \binom{i}{k}$$

- $f_i = (v_{i-1} - v_i) + (r_{i-1} - r_i) - (m_{i-1} - m_i)$

F. Alternative Platforms

$$F(k) = \sum_{i=1}^{n} f_i \binom{i}{k}$$

- $f_i = (v_{i-1} - v_i) + (r_{i-1} - r_i) - (m_{i-1} - m_i)$
- f_i does not depend on k

F. Alternative Platforms

$$F(k) = \sum_{i=1}^{n} f_i \binom{i}{k}$$

- $f_i = (v_{i-1} - v_i) + (r_{i-1} - r_i) - (m_{i-1} - m_i)$
- f_i does not depend on k

$$F(k) = \frac{1}{k!} \sum_{i=1}^{n} \frac{f_i \cdot i!}{(i-k)!}$$

F. Alternative Platforms

$$F(k) = \frac{1}{k!} \sum_{i=1}^{n} \frac{f_i \cdot i!}{(i-k)!}$$

F. Alternative Platforms

$$F(k) = \frac{1}{k!} \sum_{i=1}^{n} \frac{f_i \cdot i!}{(i-k)!}$$

- `FFT (NTT)`

F. Alternative Platforms

$$F(k) = \frac{1}{k!} \sum_{i=1}^{n} \frac{f_i \cdot i!}{(i-k)!}$$

- `FFT (NTT)`
- $a = [f_1 1!, f_2 2!, \ldots, f_n n!, 0, \ldots]$

F. Alternative Platforms

$$F(k) = \frac{1}{k!}\sum_{i=1}^{n} \frac{f_i \cdot i!}{(i-k)!}$$

- FFT (NTT)
- $a = [f_1 1!, f_2 2!, \ldots, f_n n!, 0, \ldots]$
- $b = [\frac{1}{n!}, \frac{1}{(n-1)!}, \ldots, \frac{1}{0!}, 0, \ldots]$

F. Alternative Platforms

$$F(k) = \frac{1}{k!} \sum_{i=1}^{n} \frac{f_i \cdot i!}{(i-k)!}$$

- FFT (NTT)
- $a = [f_1 1!, f_2 2!, \ldots, f_n n!, 0, \ldots]$
- $b = [\frac{1}{n!}, \frac{1}{(n-1)!}, \ldots, \frac{1}{0!}, 0, \ldots]$

$$F(k) = \frac{1}{k!}(a \times b)[n-1+k]$$

F. Alternative Platforms

$$F(k) = \frac{1}{k!} \sum_{i=1}^{n} \frac{f_i \cdot i!}{(i-k)!}$$

- FFT (NTT)
- $a = [f_1 1!, f_2 2!, \ldots, f_n n!, 0, \ldots]$
- $b = [\frac{1}{n!}, \frac{1}{(n-1)!}, \ldots, \frac{1}{0!}, 0, \ldots]$

$$F(k) = \frac{1}{k!}(a \times b)[n-1+k]$$

Time complexity: $O(n \log n)$

H. Galactic Council

The formal statement: there is an array where initially all values are equal to 0. We perform m operations: increase by 1 an element that is not the leftmost maximum. After each operation, there is a requirement for which position the leftmost maximum should be.

H. Galactic Council

- The first query definitely increases the element that should be the first maximum by 1

H. Galactic Council

- The first query definitely increases the element that should be the first maximum by 1
- For each subsequent query, there are two cases

H. Galactic Council

- The first query definitely increases the element that should be the first maximum by 1
- For each subsequent query, there are two cases
- If the requirement for the position is for the same element, then its value remains the same

H. Galactic Council

- The first query definitely increases the element that should be the first maximum by 1
- For each subsequent query, there are two cases
- If the requirement for the position is for the same element, then its value remains the same
- Otherwise, the element that becomes the maximum becomes it during this operation

H. Galactic Council

- The first query definitely increases the element that should be the first maximum by 1
- For each subsequent query, there are two cases
- If the requirement for the position is for the same element, then its value remains the same
- Otherwise, the element that becomes the maximum becomes it during this operation
- In any case, we know the value of the maximum

H. Galactic Council

- If we know the position of the maximum and its value, then all elements to the left of it must be less, and to the right all elements are less than or equal to the maximum

H. Galactic Council

- If we know the position of the maximum and its value, then all elements to the left of it must be less, and to the right all elements are less than or equal to the maximum
- For each element and each operation, there is a lower and upper bound on it after that operations

H. Galactic Council

- If we know the position of the maximum and its value, then all elements to the left of it must be less, and to the right all elements are less than or equal to the maximum
- For each element and each operation, there is a lower and upper bound on it after that operations
- There are costs for performing operations, the constraints are small — let's try to build a minimum cost flow

H. Galactic Council

- In the network, there will be a vertex for each pair “element-day”. From every such vertex, there will be an edge to the corresponding vertex for the next day (or to the sink if it is the last day)

H. Galactic Council

- In the network, there will be a vertex for each pair "element-day". From every such vertex, there will be an edge to the corresponding vertex for the next day (or to the sink if it is the last day)
- This edge will have a lower and upper flow limit corresponding to the lower and upper limit on the element

H. Galactic Council

- In the network, there will be a vertex for each pair “element-day”. From every such vertex, there will be an edge to the corresponding vertex for the next day (or to the sink if it is the last day)
- This edge will have a lower and upper flow limit corresponding to the lower and upper limit on the element
- For each day, we will also create a vertex, from which there will be an edge from the source with a capacity of 1. From this vertex to all elements edges with a capacity of 1 and cost equal to the negative cost of applying to this element on this day

H. Galactic Council

- In the network, there will be a vertex for each pair “element-day”. From every such vertex, there will be an edge to the corresponding vertex for the next day (or to the sink if it is the last day)
- This edge will have a lower and upper flow limit corresponding to the lower and upper limit on the element
- For each day, we will also create a vertex, from which there will be an edge from the source with a capacity of 1. From this vertex to all elements edges with a capacity of 1 and cost equal to the negative cost of applying to this element on this day
- To model lower bounds on flows along some edges, you can set split them into two, with one having capacity equal to the lower bound and cost equal to minus infinity

H. Galactic Council

- In the network, there will be a vertex for each pair "element-day". From every such vertex, there will be an edge to the corresponding vertex for the next day (or to the sink if it is the last day)
- This edge will have a lower and upper flow limit corresponding to the lower and upper limit on the element
- For each day, we will also create a vertex, from which there will be an edge from the source with a capacity of 1. From this vertex to all elements edges with a capacity of 1 and cost equal to the negative cost of applying to this element on this day
- To model lower bounds on flows along some edges, you can set split them into two, with one having capacity equal to the lower bound and cost equal to minus infinity
- The flow size in such a network is m, the number of vertices is $O(nm)$, the number of edges is $O(nm)$, so even without potentials it works in $O(n^2 m^3)$