ICPC 2024–2025 Northwestern Russia Qualification — Tutorial

ITMO, SPb SU, PetrSU, MAU, Online

2024-10-27

Outline

Chips

- The fastest way to try to eat from an empty can is to empty a can and then reach it! It requires $n + 1$ minutes.

Chips

- The fastest way to try to eat from an empty can is to empty a can and then reach it!
  It requires $n + 1$ minutes.
- The longest way is to eat everything first and then take any can.
  It uses $kn + 1$ minutes.

Outline

Application List

- Create a char table: 26 cells, initialized with a dot in each.

Application List

- Create a char table: 26 cells, initialized with a dot in each.
- Iterate through all the programs and mark all the first letters in the corresponding cell of the table.

Application List

- Create a char table: 26 cells, initialized with a dot in each.
- Iterate through all the programs and mark all the first letters in the corresponding cell of the table.
- Output the 26 cells of the table in 5 rows.

Outline

Ordinal Number

Plenty of ways to approach:

- parse the expression...

Ordinal Number

Plenty of ways to approach:

- parse the expression...
- check length of the string: $f(1) = 4$, $f(k) = 2f(k-1) + 1$ (with special case $f(0) = 2$)
- count the total number of “,”: $c(k) = 2^{k-1} - 1$ (with special case $n = 0$)
- count something, then use $\log_2$ of it to find the answer

Ordinal Number

Plenty of ways to approach:

- parse the expression...
- check length of the string: $f(1) = 4$, $f(k) = 2f(k-1) + 1$ (with special case $f(0) = 2$)
- count the total number of “,”: $c(k) = 2^{k-1} - 1$ (with special case $n = 0$)
- count something, then use $\log_2$ of it to find the answer
- track the bracket balance, then count “,” in the outermost set (with special case $n = 0$)
- track the bracket balance, then count “{“ in the outermost set
- track the bracket balance, then find maximum balance

Ordinal Number

Plenty of ways to approach:

- parse the expression...
- check length of the string: $f(1) = 4$, $f(k) = 2f(k-1) + 1$ (with special case $f(0) = 2$)
- count the total number of “,”: $c(k) = 2^{k-1} - 1$ (with special case $n = 0$)
- count something, then use $\log_2$ of it to find the answer
- track the bracket balance, then count “,” in the outermost set (with special case $n = 0$)
- track the bracket balance, then count “{“ in the outermost set
- track the bracket balance, then find maximum balance
- ...

Outline

RACI

- Since the role of A must be present and unique for each task, check that each row contains exactly one letter A.

Outline

Triangle on the Axis

$$\text{Area of the triangle} = \frac{\text{Base} \times \text{Height}}{2}$$

Triangle on the Axis

$$\text{Area of the triangle} = \frac{\text{Base} \times \text{Height}}{2}$$

- Let's assume that the base is the side lying on the Ox, and the height is the absolute value of the y-coordinate of the third vertex.

Triangle on the Axis

$$\text{Area of the triangle} = \frac{\text{Base} \times \text{Height}}{2}$$

- Let's assume that the base is the side lying on the Ox, and the height is the absolute value of the y-coordinate of the third vertex.
- Then we will look for the first two vertices as the leftmost and rightmost (on Ox), and the third as the vertex with the largest y-coordinate by absolute value.

Outline

Faulty Traffic Light

- There are two solutions. The first one is to implement the process described in the problem statement. Basically any implementation will work within the time and memory limits.

Faulty Traffic Light

- There are two solutions. The first one is to implement the process described in the problem statement. Basically any implementation will work within the time and memory limits.
- There is another approach that requires almost no coding but spends more time analyzing the problem on paper.

Faulty Traffic Light

- There are two solutions. The first one is to implement the process described in the problem statement. Basically any implementation will work within the time and memory limits.
- There is another approach that requires almost no coding but spends more time analyzing the problem on paper.
- Intuitively, the answer is always very small. Surely, it is below 10. One may notice that, in all four samples, the answer is either 0 or 1.

Faulty Traffic Light

- There are two solutions. The first one is to implement the process described in the problem statement. Basically any implementation will work within the time and memory limits.
- There is another approach that requires almost no coding but spends more time analyzing the problem on paper.
- Intuitively, the answer is always very small. Surely, it is below 10. One may notice that, in all four samples, the answer is either 0 or 1.
- As it turns out, the answer is always either 0 or 1. Moreover, it is 1 if and only if the second string in the input is “1101111”.

Faulty Traffic Light

- There are two solutions. The first one is to implement the process described in the problem statement. Basically any implementation will work within the time and memory limits.
- There is another approach that requires almost no coding but spends more time analyzing the problem on paper.
- Intuitively, the answer is always very small. Surely, it is below 10. One may notice that, in all four samples, the answer is either 0 or 1.
- As it turns out, the answer is always either 0 or 1. Moreover, it is 1 if and only if the second string in the input is “1101111”.
- It turns out that “1101111” is the only case when something interesting can even happen with the second digit of the number. Moreover, “1101111” in the second digit ensures that 9 and 8 always look like 5 and 6 (because leading zeroes are not displayed).

Outline

Programmers and Stones

- If all the sizes of piles are even, person whose turn is now loses. Why? Because if they take a stone from some piles, the second person may take a stone from the same piles, and continue playing this way.

Programmers and Stones

- If all the sizes of piles are even, person whose turn is now loses. Why? Because if they take a stone from some piles, the second person may take a stone from the same piles, and continue playing this way.
- Conversely, if there are some odd piles, person whose turn is now wins. How? They just take a stone from every pile of odd size.

Programmers and Stones

- If all the sizes of piles are even, person whose turn is now loses. Why? Because if they take a stone from some piles, the second person may take a stone from the same piles, and continue playing this way.
- Conversely, if there are some odd piles, person whose turn is now wins. How? They just take a stone from every pile of odd size.
- We just need to check if there is any odd number among a_i. Time complexity is $O(n)$.

Outline

Minimization by Swaps

- We try to place 1 in the first position; if we succeed, we do it.

Minimization by Swaps

- We try to place 1 in the first position; if we succeed, we do it.
- Otherwise, we try to place 2 in the first position; if we succeed, we do it.

Minimization by Swaps

- We try to place 1 in the first position; if we succeed, we do it.
- Otherwise, we try to place 2 in the first position; if we succeed, we do it.
- And so on up to 9.

Minimization by Swaps

- We try to place 1 in the first position; if we succeed, we do it.
- Otherwise, we try to place 2 in the first position; if we succeed, we do it.
- And so on up to 9.
- Then we move to the second position and the digit at that position.

Minimization by Swaps

- We try to place 1 in the first position; if we succeed, we do it.
- Otherwise, we try to place 2 in the first position; if we succeed, we do it.
- And so on up to 9.
- Then we move to the second position and the digit at that position.
- If we use a Fenwick tree or a segment tree to count the number of used digits in the segment, the complexity will be $O(n \ln(n))$.

Outline

Modular Taxi

- You are given an array and a pointer

Modular Taxi

- You are given an array and a pointer
- It is allowed to move the pointer to new place if all numbers between the old and the new positions are congruent modulo some number greater than one

Modular Taxi

- You are given an array and a pointer
- It is allowed to move the pointer to new place if all numbers between the old and the new positions are congruent modulo some number greater than one
- Move the pointer to the given position in the minimum number of steps

Modular Taxi

- Notice we never need to move away from the target position of the pointer

Modular Taxi

- Notice we never need to move away from the target position of the pointer
- Greedy approach

Modular Taxi

- Notice we never need to move away from the target position of the pointer
- Greedy approach
- Notice that the set of all taxis that can get from i to j is the set of common divisors of $a_k - a_i$, $k \in [i; j]$

Modular Taxi

- Notice we never need to move away from the target position of the pointer
- Greedy approach
- Notice that the set of all taxis that can get from i to j is the set of common divisors of $a_k - a_i$, $k \in [i; j]$
- Let us calculate the gcd of such numbers $a_k - a_i$ until it becomes equal to 1 — at this point we have to switch to start a new jump

Modular Taxi

- Notice we never need to move away from the target position of the pointer
- Greedy approach
- Notice that the set of all taxis that can get from i to j is the set of common divisors of $a_k - a_i$, $k \in [i; j]$
- Let us calculate the gcd of such numbers $a_k - a_i$ until it becomes equal to 1 — at this point we have to switch to start a new jump
- If $|a_i - a_{i+1}| = 1$, this is an impassable obstacle (the only one)

Outline

Electrician

- Let t denote the answer. Consider all the times we flip switches.

Electrician

- Let t denote the answer. Consider all the times we flip switches.
- The final flip at moment t is aimed to turn on internet anywhere and does not convey any information, so we can ignore it.

Electrician

- Let t denote the answer. Consider all the times we flip switches.
- The final flip at moment t is aimed to turn on internet anywhere and does not convey any information, so we can ignore it.
- We can distinguish two candidate houses if sequences of flips on these houses differ.

Electrician

- Let t denote the answer. Consider all the times we flip switches.
- The final flip at moment t is aimed to turn on internet anywhere and does not convey any information, so we can ignore it.
- We can distinguish two candidate houses if sequences of flips on these houses differ.
- We can not have more than $2k$ flips per house and there are t moments to make them.

Electrician

- Let t denote the answer. Consider all the times we flip switches.
- The final flip at moment t is aimed to turn on internet anywhere and does not convey any information, so we can ignore it.
- We can distinguish two candidate houses if sequences of flips on these houses differ.
- We can not have more than $2k$ flips per house and there are t moments to make them.
- Thus, $n \le \sum_{j \le 2k} \binom{t}{j}$. Our real task is to find the smallest t that satisfies this condition.

Electrician

- Let t denote the answer. Consider all the times we flip switches.
- The final flip at moment t is aimed to turn on internet anywhere and does not convey any information, so we can ignore it.
- We can distinguish two candidate houses if sequences of flips on these houses differ.
- We can not have more than $2k$ flips per house and there are t moments to make them.
- Thus, $n \leq \sum_{j \leq 2k} \binom{t}{j}$. Our real task is to find the smallest t that satisfies this condition.
- It could be done via binary search.

Electrician

- Let t denote the answer. Consider all the times we flip switches.
- The final flip at moment t is aimed to turn on internet anywhere and does not convey any information, so we can ignore it.
- We can distinguish two candidate houses if sequences of flips on these houses differ.
- We can not have more than $2k$ flips per house and there are t moments to make them.
- Thus, $n \le \sum_{j \le 2k} \binom{t}{j}$. Our real task is to find the smallest t that satisfies this condition.
- It could be done via binary search.
- Let $f(t,k) := \sum_{j \le 2k} \binom{t}{j}$. The values $f(t,1)$ are computable in constant time.
  For bigger k, it is useful to precompute the answers for inputs such that $f(t,k) \le 10^{18}$.

Outline

Wooden Matrix

- Imagine we know that some vertex v is the leaf of the tree. How to check it and find its adjacent vertex u?

Wooden Matrix

- Imagine we know that some vertex v is the leaf of the tree. How to check it and find its adjacent vertex u?
- u is the vertex with smallest $d_{v,u}$. Then we need to check that, for all other vertices w, $d_{v,w} = d_{u,w} + d_{v,u}$.

Wooden Matrix

- Imagine we know that some vertex v is the leaf of the tree. How to check it and find its adjacent vertex u?
- u is the vertex with smallest $d_{v,u}$. Then we need to check that, for all other vertices w, $d_{v,w} = d_{u,w} + d_{v,u}$.
- How to find a leaf? We may take any vertex v and then find a vertex u with the largest $d_{v,u}$. The vertex u is necessarily a leaf then.

Wooden Matrix

- Imagine we know that some vertex v is the leaf of the tree. How to check it and find its adjacent vertex u?
- u is the vertex with smallest $d_{v,u}$. Then we need to check that, for all other vertices w, $d_{v,w} = d_{u,w} + d_{v,u}$.
- How to find a leaf? We may take any vertex v and then find a vertex u with the largest $d_{v,u}$. The vertex u is necessarily a leaf then.
- After that, we may drop this leaf and continue this process for other vertices. The time complexity is $O(n^2)$.

Wooden Matrix

- Imagine we know that some vertex v is the leaf of the tree. How to check it and find its adjacent vertex u?
- u is the vertex with smallest $d_{v,u}$. Then we need to check that, for all other vertices w, $d_{v,w} = d_{u,w} + d_{v,u}$.
- How to find a leaf? We may take any vertex v and then find a vertex u with the largest $d_{v,u}$. The vertex u is necessarily a leaf then.
- After that, we may drop this leaf and continue this process for other vertices. The time complexity is $O(n^2)$.
- Alternatively, we may find a minimal spanning tree, and then note that this tree should be exactly the tree in which we calculated the distances.

Outline

Count the Operations

Let us try simulation first:

- for each value of `i`:
- for each `if`:
- process it
- way too slow

What are we lacking?

Count the Operations

Let us try simulation first:

- for each value of `i`:
- for each `if`:
- process it
- way too slow

What are we lacking?

- find the next value of `i` when something happens, fast
- find the next `if` where something happens, fast

Count the Operations

Now, consider a faster simulation:

- consider `ifs` as pairs (x_j, line number)
- store these pairs in a set
- to find the next event, we have to consider the `upper_bound` of the current position
- we can count the number of operations from (i, old line) to (x_j, new line) in $O(1)$

Turns out this is already fast enough. Why?

Count the Operations

Now, consider a faster simulation:

- consider `ifs` as pairs (x_j, line number)
- store these pairs in a set
- to find the next event, we have to consider the `upper_bound` of the current position
- we can count the number of operations from (i, old line) to (x_j, new line) in $O(1)$

Turns out this is already fast enough. Why?

- Lemma: if we enter some `if` twice, we got into an infinite loop
- indeed, after we execute the `if` body, everything will be exactly as before
- so, we enter each `if` at most once, or detect an infinite loop

Outline

Balls of Three Colors

- The intended solutions work in $O(r + g + b)$ time. The simplest linear solution is based on the following idea.

Balls of Three Colors

- The intended solutions work in $O(r + g + b)$ time. The simplest linear solution is based on the following idea.
- Red balls split the whole line into either $r + 1$, r, or $r - 1$ segments, with each such segment consisting of alternating blue and green balls.

Balls of Three Colors

- The intended solutions work in $O(r + g + b)$ time. The simplest linear solution is based on the following idea.
- Red balls split the whole line into either $r + 1$, r, or $r - 1$ segments, with each such segment consisting of alternating blue and green balls.
- Segments of even length do not contribute to $g - b$ and can be filled in two different ways.

Balls of Three Colors

- The intended solutions work in $O(r + g + b)$ time. The simplest linear solution is based on the following idea.
- Red balls split the whole line into either $r + 1$, r, or $r - 1$ segments, with each such segment consisting of alternating blue and green balls.
- Segments of even length do not contribute to $g - b$ and can be filled in two different ways.
- Each segment of odd length either increases $g - b$ or decreases it. Enumerate the number of odd segments.

Balls of Three Colors

- The intended solutions work in $O(r + g + b)$ time. The simplest linear solution is based on the following idea.
- Red balls split the whole line into either $r + 1$, r, or $r - 1$ segments, with each such segment consisting of alternating blue and green balls.
- Segments of even length do not contribute to $g - b$ and can be filled in two different ways.
- Each segment of odd length either increases $g - b$ or decreases it. Enumerate the number of odd segments.
- Now, we know the numbers of even segments, "green" odd segments (with more green balls than blue balls) and "blue" odd segments.

Balls of Three Colors

- The intended solutions work in $O(r + g + b)$ time. The simplest linear solution is based on the following idea.
- Red balls split the whole line into either $r + 1$, r, or $r - 1$ segments, with each such segment consisting of alternating blue and green balls.
- Segments of even length do not contribute to $g - b$ and can be filled in two different ways.
- Each segment of odd length either increases $g - b$ or decreases it. Enumerate the number of odd segments.
- Now, we know the numbers of even segments, “green” odd segments (with more green balls than blue balls) and “blue” odd segments.
- Finally, we need to choose the placements and the lengths of the segments.

Balls of Three Colors

- Distributing the types of segments is a product of binomial coefficients.

Balls of Three Colors

- Distributing the types of segments is a product of binomial coefficients.
- Distributing the lengths is more complicated. But we have already distributed the types.

Balls of Three Colors

- Distributing the types of segments is a product of binomial coefficients.
- Distributing the lengths is more complicated. But we have already distributed the types.
- Now we have some extra “charges” that allow us to extend the length of some segments by two. Any number of charges can be applied to each given segment.

Balls of Three Colors

- Distributing the types of segments is a product of binomial coefficients.
- Distributing the lengths is more complicated. But we have already distributed the types.
- Now we have some extra "charges" that allow us to extend the length of some segments by two. Any number of charges can be applied to each given segment.
- This is the famous "combinations with repetitions" problem. The answer is also a binomial coefficient.

Balls of Three Colors

- Distributing the types of segments is a product of binomial coefficients.
- Distributing the lengths is more complicated. But we have already distributed the types.
- Now we have some extra “charges” that allow us to extend the length of some segments by two. Any number of charges can be applied to each given segment.
- This is the famous “combinations with repetitions” problem. The answer is also a binomial coefficient.
- Alternative solution. There are several ways to prove the following formula: $f(a, b, c) = f(a-1, b-1, c) + f(a-1, b, c-1) + f(a, b-1, c-1) + 2f(a-1, b-1, c-1)$, where $f(r, g, b)$ is the answer to the problem. This also leads to a linear solution.

Outline

Definitely Not Chess

- Retrograde analysis.

Definitely Not Chess

- Retrograde analysis.
- Pieces move symmetrically, so there is no need to store the graph of all moves explicitly.

Definitely Not Chess

- Retrograde analysis.
- Pieces move symmetrically, so there is no need to store the graph of all moves explicitly.
- We can always rotate the board so that the white king would be in the bottom left quarter of the board. It makes the search space 4 times smaller.

Definitely Not Chess

- Retrograde analysis.
- Pieces move symmetrically, so there is no need to store the graph of all moves explicitly.
- We can always rotate the board so that the white king would be in the bottom left quarter of the board. It makes the search space 4 times smaller.
- Additionally, we can use symmetry over the diagonal to get rid of almost a half of the remaining positions.

Outline

Problem Authors

Ivan Bochkov

Nikolay Dubchuk

Nikita Gaevoy

Anastasia Grigorieva

Mikhail Ivanov

Ivan Kazmenko

Anton Maidel

Vladislav Makarov