RCPCamp Day 2 - Editorial

October 2023

1 Special Thanks

This contest couldn't have been possible without:

- Ștefan Cosmin Dăscălescu, Andrei Paul Iorgulescu, Luca Valentin Mureșan, Andrei Chertes, Casian Pătrășcanu and David-Ioan Curcă, the people who authored and prepared the problems.

- Andrei Robert Ion, Bogdan Ioan Popa, Andrei Boacă, Alexandru Gheorghieș and Tiberiu Ștefan Cozma for testing the round and providing valuable feedback.

- Alexandru Vasiluță, developer of kilonova, platform on which the contest was held on.

- Marius Dumitran and Paul Diac for making sure all requirements are met for the contest to take place optimally.

2 Problem Maximum Distance

AUTHOR: LUCA VALENTIN MUREȘAN

To solve the problem, we must notice that at least one of the two subarrays will end on the position n_{th} position.

To prove this claim, suppose the optimal solution uses subarrays l_1 r_1 and l_2 r_2 respectively, such that $r_1 < n$ and $r_2 < n$. Then, a valid solution would be to increase both r_1 and r_2 by 1, and, of course, the answer can only increase.

Thus, we can take two cases: the subarray in the first array finishes on position n and the subarray in the second array finishes on position n, respectively. Now, we can set the right end of the other array at any position, and iterate through the left ends decreasingly, keeping the current distance, and trying it for our answer. The time complexity will be $O(N^2)$.

3 Problem The floor is lava

AUTHOR: LUCA VALENTIN MUREȘAN

Let's try to look at one of the K people. We can notice that we can reach any point in the room with at most $n + m$ moves.

We can fix r, the number of moves we make. By making at most r moves we can reach in a *diamond* of radius r centred in the starting position. If we make at most r steps, it's obvious that we want to get as high as possible. That is, we want to get to the highest element in that diamond.

To do this, we can use range minimum queries on the diagonals of a matrix. To ease the implementation, we can rotate the matrix by 45 degrees, and instead of having to do range minimum queries on diagonals, we can do range minimum queries on rows and columns.

Time complexity: $O(n^2 \log_2 n + K \cdot n)$

4 Problem Basketball

AUTHOR: ȘTEFAN COSMIN DĂSCĂLESCU

First of all, if $n = 1$, we have no solutions and have to output -1.

If n is a multiple of 3, it is optimal to use only 3-throws, since they guarantee reaching the desired result as quickly as possible.

If n is congruent to 1 modulo 3, we will need at least two 2-throws to match the result. After using two 2-throws, we will be left with having to minimize the number of throws to reach $n - 4$, which is a multiple of 3, so we can apply the idea at case 1.

If n is congruent to 2 modulo 3, we will need to use at least one 2-throw (else, the result we would get would be a multiple of 3). After using one 2-throw, we will reach the number $n - 2$, which is a multiple of 3, so we can apply the idea at case 1.

Time complexity: $O(1)$

5 Problem Edenland

AUTHOR: ANDREI PAUL IORGULESCU

To solve this problem, we must really understand how going on a track works. At the beginning of it, without any exception, Bob will wait for Alice. Then, after some time of going without a pause, he will again have to wait. He will do this several times, until he finishes the track.

Now, let's notice something: for some position p, if we know that Bob waits at position p, there is a predetermined next position where Bob will wait for Alice. Let's call that $nxt[p]$. We can suppose that $nxt[p] = n + 1$ if Bob does not wait again after position p.

If we can precalculate the values of nxt, then we can solve this problem by binary lifting: remember, for every position p, the ith waiting time after position p for every i power of 2 less than n. To solve a query, we can find the last time Bob waits before position r (let that be position k), and then compute the difference of arrivals time with prefix sums on a and b, as the sum of b from k to r minus the sum of a from $k + 1$ to r.

Now, we are left with finding the values of nxt for every position p. Let's notice that Bob will start moving at the moment Alice has started game $p + 1$. Suppose k is the first position Bob will wait again. That means that the sum of a from $p + 1$ to k must be greater than the sum of b from p to $k - 1$. Let $pa[i]$ be the prefix sum of a to position i, and $pb[i]$ the prefix sum of b to position i.

Rewrite the formula:

$$pa[k] - pa[p] > pb[k - 1] - pb[p - 1] \Leftrightarrow pa[k] - pb[k - 1] > pa[p] - pb[p - 1]$$

If we consider a new array, $dif[i] = pa[i] - pb[i - 1]$, we can say that k satisfies the property that it is the first position greater than p such that $dif[k] > dif[p]$. This can be easily calculate, for example, using a stack in $O(N)$, as it is a very classical problem called "next greater element".

Time complexity: $O(NlogN + QlogN)$

6 Problem Fiboxor

Author: Andrei Paul Iorgulescu

First of all, we can compute the sum of $f[i]$ between l and r as the sum of $f[i]$ for $i <= r$ minus the sum of $f[i]$ for $i < l$.

To compute the sum of $f[i]$ for $1 \leq i \leq k$, we must first prove something: $f[i] = 2^{(i-1)/2}$. The demonstration is quite simple, inductively:

- The base cases, i = 1 and i = 2 are trivial.

- For $i >= 3$, i is congruent to 1 modulo 2, $f[i-1] = f[i-2] = 2^{(i-3)/2}$, so $f[i] = 0 \oplus 2^{(i-3)/2+1} = 2^{(i-1)/2}$.

- For $i >= 4$, i is congruent 0 modulo 2, $f[i-1] = 2^{(i-1)/2}$ and $f[i] = 2^{(i-3)/2}$, so $f[i] = 2^{(i-3)/2} \oplus (2^{(i-1)/2} + 2^{(i-3)/2}) = 2^{(i-1)/2}$.

Thus, we basically need to be able to compute $2^0 + 2^0 + 2^1 + ... + 2^{(k-1)/2}$. If k is not divisible by 2, we can add $2^{(k-1)/2}$ separately and assume k is even. If k

3

is even, we actually have to compute $2 * (2^0 + 2^1 + ... + 2^{k/2-1})$. That sum is equal cu $2 * (2^{k/2} - 1)$, which we can calculate with logarithmic exponentiation.

The time complexity will be $O(logMAX)$ per testcase, where MAX is the maximum possible value of r.

7 Problem Suceava

AUTHOR: ANDREI CHERTES

First, we can notice that we can solve the problem for each gang independently. The graph for each gang is a fores and during the T days an edge may be either added or removed from the forest. For each query we need to find the maximum diameter from each tree in this dynamic forest. Now we can solve this problem using dynamic connectivity technique, storing in disjoint set union data structure the two endpoints of the diameter.

The editorial is not complete yet

8 Problem Minimize Sum

AUTOR: LUCA VALENTIN MUREȘAN

First, we can notice that the n_{th} element in the array (a_n) will never be added to T. Now, we will only use the first $n - 1$ elements.

It's easy to see that we will add all a_i with i from 1 to $n - 1$ except one of them. Since we want to minimize T we would like to exclude the maximum element.

To do this, let's say that p is the position of the maximum element in the array. Then, we can push p at the front of the deque and all the other elements at the end of the deque.

Also, a greedy solution where we just push a_i to the front if $deque.front$ is less than $deque.back$ and to the back otherwise would work because of the proof above.

Time complexity: $O(n)$

9 Problem AI Thoughts

AUTHOR: ANDREI PAUL IORGULESCU

Let's do dynamic programming. Let dp[i][0/1/2/3/4] be the maximum length to form the thought until position i(0), eventually +-x +- y where (x,y) are the coordonates of the final chosen point.

The reason why it works to try all transitions is that —x— = max(x,-x), so the Manhattan distance between two points p and q is maximum when we consider the signs of differences correctly.

Now, let's consider the transitions. For instance, for ending on +x,+y(dp[i][1]), we have the following cases:

- If we come from dp[i - 1][1], it will be dp[i - 1][1] - x - y + x + y, which is equal to dp[i - 1][1].

- If we come from dp[i - 1][2], it will be dp[i - 1][2] - x + y + x + y, and, as we want to maximize it, we want to maximize +y.

- If we come from dp[i - 1][3], it will be dp[i - 1][3] + x - y + x + y, and, as we want to maximize it, we want to maximize +x.

- If we cone from dp[i - 1][4], it will be dp[i - 1][4] + 2 * (x + y), so we want to maximize x + y.

Writing all transitions, we notice that we only care, for every value, about the neurons which have one of the following properties: maximum x, maximum -x, maximum y, maximum -y, maximum x + y, maximum x - y, maximum -x + y, maximum -x - y.

Thus, we can retain this 8 neurons for every value, and do the dp above, or just a simpler dp d[i][j] = the maximum length to generate the thought consisting of the first i values and ending in the j-th of the 8 candidate neurons. We can write the transitions in 8 * 8.

The complexity will be $O(n + sum(m))$ with a high constant factor.

10 Problem KSumtT

AUTOR: LUCA VALENTIN MUREȘAN

If we have $a_1 \cdot a_2 \cdot \cdots \cdot a_t = a_2 \cdot a_3 \cdot \cdots \cdot a_{t+1}$, then we have $a_1 = a_{t+1}$. In the same way, we get that $a_1 = a_{t+1} = a_{2 \cdot t+1} = a_{3 \cdot t+1} = \dots$;
$a_2 = a_{t+2} = a_{2 \cdot t+2} = \dots$; $a_3 = a_{t+3} = a_{2 \cdot t+3} = \dots$

We can imagine this equalities as groups. Let $b_0 = a_1$ (the value of the first group), $b_1 = a_2$ (the value of the secound group), $b_{i-1} = a_i$ (the value of the ith group, $1 \leq t$). Then, $a_1 + a_2 + a_3 + \cdots + a_t = S \Leftrightarrow$
$b_0 \cdot (x+1) + b_1 \cdot (x+1) + \cdots + b_p \cdot (x+1) + b_{p+1} \cdot x + b_{p+2} \cdot x + \cdots + b_{t-1} \cdot x = S$.
Where $x = \frac{k}{t}$ (the number of times, all groups will appear at least once) and $p = k \mod t$ (the first p groups will appear $x + 1$ times, the other x times).

Now the equality writtten aboves becomes
$(x + 1) \cdot (b_0 + b_1 + \cdots + b_p) + x \cdot (b_{p+1} + b_{p+2} + \cdots + b_{t-1}) = S$.

Now, we can fix $a = b_0 + b_1 + \ldots b_p$, then $b_{p+1} + b_{p+2} + \cdots + b_{t-1} = \frac{S - a \cdot (x+1)}{x}$. So we reduced the problem to finding the number of ways to write a number a as sum of $p + 1$ numbers greater than 0 (similar for the other side). This is a classic stars and bars problem.

11   Problem Parallelogram

Author: Luca Valentin Mureşan

We know that a quadrilateral is a paralellogram if and only if two pairs of opposite sides are equal in length. Thus, we have two cases:

1. There are two lengths $x \neq y$ such that both x and y appear at least twice.

2. There is one length x such that x appears at least four times.

We can easily check these by making a frequency array.

12   Problem Blabla

Author: Andrei Paul Iorgulescu, Luca Valentin Mureşan

To solve this problem, we can use divide and conquer. To solve for the interval $[l, r]$, we can add the number of subarrays included in $[l, mid]$ and in $[mid + 1, r]$, and we are left with calculating the number of good subarrays with $l <= mid$ and $mid + 1 <= r$, where mid is the middle of $[l, r]$.

To do this, we can take the following four cases: both the minimum and maximum of the subarray are on the left side, both are on the right side, the minimum is on the left and the maximum on the right, the maximum is on the left side and the minimum on the right side.

Let's treat the first to cases first, since they are easier. We will present the solution for the first case, since the second one is symmetrical. We can iterate over the left end, st, from mid to l. Since we can find the maximal and minimal value on the suffix of the left side starting from st, we will need dr to respect the following properties:

- The maximum on the prefix of the right side until dr is less than or equal to the maximum on the left side.

- The minimum on the prefix of the right side until dr is greater than or equal to the minimum on the left side.

- $dr <= mx - mn + st$ where mx and mn are the maximums and minimums respectively.

Clearly, only a prefix of dr will work, and to find it we can use two pointers or binary search, making the complexity of this part $O(nlog)$ or $O(nlog^2)$.

Case 2 is similar, however do note that you must not overcount or undercount. An idea used for this in the main solution is to consider the minimum as the leftmost element with the minimum value, and the maximum - the leftmost element with maximum value.

Now, let's take the third and fourth cases. We will focus on the third one, since they are, again, pretty much the same. Again, let's iterate over st from mid to l. We know we want the minimum to be on the left side, which bounds that dr is lest than or equal to a value which you can find by two pointers or binary search. We also want the maximum to be on the right side, which indicates that dr is greater than or equal to some value which, again, we can binary search or find by two pointers.

We can also rewrite the formula: $mx - dr <= mn - st$. We know the value of $mn - st$, and we can calculate for every dr the value of $mx - dr$ where mx is the maximum on the prefix of the right side until dr. We have reduced our problems to queries of the form: "on some interval, what is the number of elements less than or equal to some value". This is a very classical problem and can be solved by sorting the queries and values and using a binary indexed tree.

The time complexity for every divide step will be $O(nlog)$, so the total time complexity will be $O(nlog^2)$.

13   Problem Dush

First we will check for each person which timeslot could bit a good fit. We store it in a matrix $fit[i][j] = $ the jth timeslot (in order) which is a good fit for the ith person.
A clearly correct solution (although it is not fast enough to pass the time limit) would be a backtracking approach for the order of people that go take a shower. Now, if we know the order of people, suppose after the ith person the best time for that order was T time. Then for the $i + 1^{th}$ person we can just check what is the first time he/she could fit in an interval after time T. We can find this first interval by either binary searching for it in $fit[(i + 1)^{th}$ person in order].
How can we improve this solution? Notice that for a subset of people we do not really care for the order in regards to the result, we just want the best result for all orders of this subset. Therefore, we can change our backtracking into a $dp[mask] = $ the best time for any order of the people that are equal to the subset present in $mask$, where $mask$ is a binary mask representing for each person whether it is present or not (already took his shower). Then we can just iterate the masks starting from 0 to $2^n - 1$ and iterate the next

person added to the mask (as a bit).

Author: Casian Pătrăşcanu

14   Problem Dragons

Author: Andrei Chertes

First, lets see that the heights which affect the power, P of the dragon should for a non-decreasing subset, containing all the maximum values from the path (u, v). Moreover using a simple induction we can conclude that if P reaches a value below 0 before encountering the maximum height on the path, it cannot become 0 in the end. Now, the problem is reduced to subset sum problem, with the additional condition that if P becomes 0 when there is an even number of maximum values left to process, then P will be 0 in the end. Otherwise, there is no way we can make P equal to 0.

To solve the subset sum problem on chains, we will use Mo's algorithm on trees, combined with knapsack dynamic programming which must also support erasing existing elements (this can be done easily by keeping the number of ways to reach a sum modulo a big number).

The editorial is not complete yet