All-Ukrainian Collegiate Programming Contest 2023

II stage

Very Fast Editorial
(and many exercises for the readers)

Thanks for your participation!

Problem A. Albuquerque's Avenues

Root tree from some node and consider subtree of node v with parent par. Let's look what happens when we do operation with nodes v, par:

- If v was `S` and par was `F`, then in subtree of v the number of `S` will decrease by 1
- If v was `F` and par was `S`, then in subtree of v the number of `S` will increase by 1

Note that the numbers of `S` in subtree of v never change during any other operations. From this, we can make the following observation: the number of `S` in subtree of every node v has only two values it can attain. Moreover, this value determines the direction of the edge from v to its parent.

It turns out that every configuration which satisfies these conditions is indeed reachable. The proof is left as an exercise to the reader. So, it's enough to count the number of possible final configurations of `S` and `F`, given that we know two possible values for the numbers of `S` for each subtree (these two values differ by 1, by the way). How to do it?

We can do tree dp. Let's say that for node v we know, there must be exactly x_v or $x_v + 1$ labels `S` inside its subtree. We will denote the numbers of ways to get corresponding configurations by $dp_0[v], dp_1[v]$ correspondingly.

Consider children of v: $v_1, v_2, \ldots, v_k$. For each v_i, there are $dp_0[v_i]$ ways to get x_{v_i} labels `S` in its subtree, and $dp_1[v_i]$ ways to get $x_{v_i} + 1$. How to combine all this knowledge? With polynomials.

Indeed: consider polynomial $\prod_{i=1}^{k}(dp_0[v_i] \cdot t^{x_{v_i}} + dp_1[v_i] \cdot t^{x_{v_i}+1})$. Its coefficient near t^y represents the number of ways to assign labels in subtrees so that we have exactly y labels `S`. We can calculate this polynomial in $O(k \log k^2)$ with FFT and Divide and Conquer. Then, consider possible labels of node v, and update corresponding dp values.

The total asymptotic is $O(n \log n^2)$.

Problem B. Breaking Bad

There are many ways to solve this problem.

Consider what happens when Walt is shouting. For each node in the tree that had money, it will move to the parent node (except node 0). Some money will get merged. The largest amount of money in a node can change only during one of these merges, and there are at most $n-1$ of these merges (since every time there is a merge, the number of nodes with nonzero money decreases). Let's detect all the merges.

Merge can happen in two cases: some money arrives at node 0, or for some nodes u and v with equal depth h the merge happens in their lowest common ancestor. It's easy to detect both. For the second one, for each depth h, consider all nodes of depth h in DFS order, and look at when each pair of adjacent nodes gets merged (by looking up their LCA).

Having all the merging information, we can go step by step, keeping track of the sets of nodes that are already merged, and keeping track of the largest sum of money Walt can get.

Asymptotics $O(n \log n)$ (but we can also get $O(n)$).

Problem C. Chemistry Class

Sort the numbers, now $a_1 \le a_2 \le \ldots \le a_{2n}$.

Now, if $a_{2i-1} + A < a_{2i}$ for some i, then there is no solution. Indeed: there has to be at least one pair of form (a_x, a_y) with $x \le 2i-1, y \ge 2i$, since we can't split the first $2i-1$ into pairs only among themselves. Otherwise, there is a solution: we can simply form pairs (a_{2i-1}, a_{2i}). Now, assume $a_{2i-1} + A \ge a_{2i}$ for all i.

Consider solution with the largest number of pairs with distance at most B, denote it by k. We will refer to these k pairs as B-pairs, and to other pairs as A-pairs. First, note that we can make sure that no two A pairs "intersect"as segments, the same for B. Indeed, if elements from B pairs are $a_{x_1} \le a_{x_2} \le \ldots a_{x_{2k}}$, then we can form pairs $(a_{x_{2i-1}}, a_{x_{2i}})$.

Now, consider the first $2i$ elements. There are two cases:

- All of them are split into i pairs somehow. Then, we just need to find the maximal number of B-pairs we can form there separately.
- They are split into $i-1$ pairs, with one A-element and one B-element remaining to be paired with elements with index $\ge i-1$, we refer to these pairs as "in-between pairs". Assume that when splitting only these $2i$ elements into i A-pairs and B-pairs, we can get at most k_1 B-pairs. Then we can have exactly k_1 B-pairs among these formed $i-1$ pairs in this optimal solution. Indeed: if we have less than k_1, then it's optimal to just break the in-between pairs, and pair the first $2i$ elements among themselves. This won't decrease the number of B-pairs. And, clearly, we can't have more than k_1 B-pairs. So, we will have exactly k_1 B-pairs.

  One of these leftover elements is going to be a_{2i}. Our only goal is then to make the other leftover element as rightmost as possible.

Similarly, when we consider the first $2i-1$ elements, we have all elements paired, except one leftover element. Our goal is once again to make it as rightmost as possible.

So, we can do a kind of dynamic programming. For even positions, $2i$, we can store the following values:

- The largest number of B-pairs we can form by splitting all $2i$ elements into pairs, k_1.
- If we split them into $i-1$ pairs, and a_{2i} belongs to in-between A-pair, what's the rightmost possible position of the remaining B-element (assuming the number of B-pairs is largest possible)
- If we split them into $i-1$ pairs, and a_{2i} belongs to in-between B-pair, what's the rightmost possible position of the remaining A-element (assuming the number of B-pairs is largest possible)

For odd positions, we will keep similar things. The transitions will take $O(1)$ time, and the total asymptotic is $O(n)$.

Problem D. Daily Disinfection

The operation that we can do is just the following:

- Swap adjacent `0` and `1`

We need to find the smallest number of operations required so that every position has been `0` at some point.

Here is another rephrasing of the problem. We have several `0`s. In one operation, we can move `0` to any adjacent position. We need to visit all positions with `0`s in the smallest number of operations.

The obvious lower bound is the number of `1`s. In fact, it is achievable when any of the following is true:

- The leftmost character is `0`
- The rightmost character is `0`
- There are two adjacent `0`s

Indeed, if the first condition holds, then each `0` can just "visit"the ones to the right of it (before the next `0` or the end of the string). Same for the second case. For the third case, if these zeros are at positions $i, i+1$, then all zeros at positions $\leq i$ can visit ones to the left of them, and all zeros at positions $\geq i+1$ can visit ones to the right of them.

The remaining case is trickier. It turns out that the minimal number of operations required would be equal to the number of `1`s plus the length of the shortest segment of ones. The proof is left to the reader as an exercise.

Total asymptotic: $O(n)$.

Problem E. Equalizer Ehrmantraut

First of all, if some bit is present in all a_i, it will always be present in all a_i, and if some bit is not present in some a_i, it's impossible that it's present in each a_i in the end. So, we can find $x =$`AND` of all a_i, subtract x from all a_i, and now we need to find the smallest number of operations required to make all numbers 0.

If at least one a_i is 0, we can just repeatedly choose an adjacent pair of elements, in which one is 0, and one isn't, and replace the nonzero one with 0. So, in this case, the answer would be just the number of nonzero elements.

Now suppose all elements are nonzero. The moment we get the first 0, we will need to do exactly $n-1$ extra operations. So, the answer is equal to the smallest number of operations needed to get 0, plus $n-1$.

How can we get 0? By considering some segment of length k on which `AND` is 0, and applying $k-1$ operations there. So, the answer is equal to $n-1$ plus the length of the shortest segment with `AND` equal to 0, minus 1. It remains to find the length of the shortest segment with `AND` equal to 0.

We can do this in any way you want. The simplest way is iterate i from 1 to n, and to find the rightmost j for which `AND` on segment $a[i:j]$ is nonzero. These j would be nonincreasing. We can maintain the number of times each bit appears on a segment when increasing i and j to check whether `AND` is zero.

The complexity is then $O(n \log maxA)$.

Problem F. Felina

Let's understand when multiset S satisfies this condition. Assume that the sum of integers in multiset S has form $2^k + x$, where $0 \le x < 2^k$. Note that if we can split it into two groups with the same highest bit, this bit must be $k - 1$. We will show that it can't be split into two groups with equal highest bit if and only if the following condition holds:

- There exists some bit $i \le k$, such that the sum of powers of 2 which are at least 2^i in S is at least $2^{k+1} - 2^i$

If this condition holds, then no matter how we split, at least one group will contain at least half of that sum, and, since it's divisible by 2^i, it will contain at least 2^k. So, its highest bit would be k, and the highest bits won't be equal.

We can show that this condition is true by induction by the sum of elements of S. Denote it by n. It's trivial for $n = 1$.

Suppose that it's true for all smaller values, let's show it for n. Let S be any group we can't split this way, let $n = 2^k + x$ again. If $n = 2^{k+1} - 1$, the condition holds for $i = 0$. Now, consider all powers of 2 in S larger than 2^0, and divide them by 2.

If we could split this group in the required way, we would be able to go back (multiply all powers of two by 2), and add the remaining ones into groups appropriately, and split the initial multiset. Indeed: if we have numbers a and b with equal highest bit, and need to add $n - a - b$ ones to them so that they still have equal highest bit, it's easy to do: always add 1 to the smaller number, and, unless $n = 2^{k+1} - 1$, we will get two numbers with same highest bit in the end.

So, we can't, and therefore there exists some $i \le k - 1$ for which the condition holds there. This means that this condition holds for $i + 1$ in the initial multiset.

Now let's use this observation to solve the problem. First, we can calculate $cnt[i]$ = the total number of multisets with sum i for all $1 \le i \le n$. It's just:

- $cnt[i] = cnt[i - 1]$ for odd i
- $cnt[i] = cnt[i - 1] + cnt[i/2]$ for even i

That's because we can either get multiset for i by adding 2^0 to multiset for $i - 1$, or, if there are no ones, by multiplying all elements of multiset with sum $i/2$.

Now, we can calculate the answer. The answer looks like this:

- $ans[i] = 1$ for $i = 2^k$
- $ans[i] = cnt[i]$ for $i = 2^{k+1} - 1$

  For all other i:

  $ans[i] = ans[i - 1]$ for odd i
- $ans[i] = ans[i - 1] + ans[i/2]$ for even i

That's because for these other i, we can get a multiset in which this condition holds in the same two ways: by adding 2^0 to multiset for $i - 1$, or, if there are no ones, by multiplying all elements of multiset with sum $i/2$.

Total asymptotic: $O(n)$.

Problem G. Genius Gus

Consider some operation. Consider only the positions that are changing during this operation. Note that the total number of ones and zeros doesn't change, and the positions that changed in the resulting string will always look like this: first k zeros, then k ones. So, in s they must have looked like this: first k ones, then k zeros (for some k).

Vice versa, if we have some subsequence of this form in s (first k ones, then k zeros), we can sort it, and change the values at exactly those $2k$ positions. So, the problems reduces to the following:

- Find the number of subsequences of s, which have the following form: first k ones, then k zeros, for some $k \geq 0$.

Case $k = 0$ leads to a single empty subsequence. Now, let's count the number of such subsequences with $k \geq 1$.

Fix the last occurrence of one in this subsequence. Let's say we have a ones before it, and b zeros after. Then, for each k, we need to add $\binom{a}{k-1}\binom{b}{k}$ to the answer. This already gives us $O(n^2)$ algorithm.

However, we can simplify the sum $\sum_{k=1}^{\infty}\binom{a}{k-1}\binom{b}{k}$. Note that $\binom{a}{k-1}\binom{b}{k} = \binom{a}{a-k+1}\binom{b}{k}$. So, it's same as choosing $(a-k+1)+k = a+1$ elements in total, $a-k+1$ from a, and k from b. And we are summing this over all k. Then sum is just equal to the total number of ways to choose $a+1$ elements from $a+b$ elements. This is just $\binom{a+b}{a+1}$.

So, we need to iterate over all ones and sum up all corresponding values $\binom{a+b}{a+1}$.

Total complexity: $O(n)$ (since after precalculating factorials and inverse factorials up to $O(n)$ in $O(n)$ time, we can get binomial coefficients in $O(1)$).

Problem H. Healthy Howard Hamlin

The answer is just $\lfloor \frac{A+B}{2} \rfloor$. We can achieve it by running in fast mode for the last B minutes and alternating between slow and fast (1 minute each) for the first $A - B$ minutes.

It's easy to see that we can't do better. Consider the last time Howard started running in the fast mode, let's say at time x. Then he ran in fast mode for at most $\lfloor \frac{x}{2} \rfloor$ of the first x minutes. As for last $A-x$ minutes, he can run in fast mode for at most $min(B, A - x)$. So, we can't do better than $\lfloor \frac{x}{2} \rfloor + min(B, A - x)$. If $x < A-B$, this is at most $\lfloor \frac{A-B}{2} \rfloor + B = \lfloor \frac{A+B}{2} \rfloor$, otherwise it's $\lfloor \frac{x}{2} \rfloor + A - x \leq \lfloor \frac{A-B}{2} \rfloor + A - (A-B) = \lfloor \frac{A+B}{2} \rfloor$.

Asymtotics $O(1)$ per test case.

Problem I. Immense Input

The problem asks to find the largest possible value of $a_1+a_2+\ldots+a_T$, where $a_1, a_2, \ldots, a_T$ are nonnegative integers with $a_1^2 + a_2^2 + \ldots + a_T^2 \le K$ (in the statement t might be smaller than T, and numbers have to be positive, but we can just add extra $T - t$ zeros).

Consider such array $a_1, a_2, \ldots, a_T$ with the largest possible sum, and among those consider an array for which the value of $a_1^2 + a_2^2 + \ldots + a_T^2$ is the smallest possible. Then no two values can differ by at least 2. For example, if $a_1 \le a_2 - 2$, then we can replace a_1 by $a_1 + 1$, a_2 by $a_2 - 1$. The sum won't change, the sum of squares will become $a_1^2 + a_2^2 + 2(1 + a_1 - a_2) < a_1^2 + a_2^2$. So, there is some x such that $a_i = x$ or $x + 1$ for all i.

What is the maximum possible x? Clearly, $\lfloor\sqrt{\frac{K}{T}}\rfloor$. With this x, what is the maximum possible number of a_is equal to $x + 1$? It's $\lfloor\frac{K-Tx^2}{2x+1}\rfloor$ (since replacing each x^2 with $(x + 1)^2$ "consumes"$2x + 1$). Finding these values takes $O(1)$ per test case.

Problem J. Jesse's Job

Consider the split of the permutation into cycles. If there are at least two cycles, we can color one of them in yellow, and everything else in blue. This way, we would put all elements at their places.

Assume now there is only a single cycle. Then there is no way to put all elements at their places: this would mean that blue elements form cycle(s), and yellow too. Obviously, we can't get $n-1$ elements at their places either.

It turns out, however, that we can get $n-2$ elements at their places! Indeed, swap elements 1 and 2 of the permutation. It will decompose into two cycles. Color one cycle in yellow, one in blue. All elements would fall at the correct places, except 1 and 2 that we swapped. So, it's possible to get $n-2$.

Asymptotics $O(n)$.

Problem K. Kim

If $a = b$, then $(a + x)$ `AND` $(b + x) = a + x$ for any x. So, best we can do is just choose $x = 0$.

Now assume $a \neq b$. Wlog $a < b$. Then, let's choose x so that $b + x$ is 2^k for some k. Then $a + x$ can't have any common bits with 2^k, so $(a + x)$ `AND` $(b + x) = 0$ for such x. So, find any such x.

Asymptotic: $O(1)$.

Problem L. Lalo's Lawyer Lost

For the tree, this problem is pretty well-known. The answer can be calculated as follows: for each edge, find the number of nodes on both sides and let those be x, y. Then, add $min(x, y)$ to the answer. This is clearly an upper bound: this edge will be contained in at most $min(x, y)$ paths between nodes from the same pairs. It can be shown that it's reachable: it's enough to choose a centroid, consider its subtrees, and pair nodes from different subtrees, which will always be possible.

Here, however, we have cycles. Still, we can try to use this idea. For edges that are not a part of any cycle, we can do the same: find the number of nodes on both sides: x, y, and add $min(x, y)$ to the answer. Now, let's handle the edges of the cycles.

Consider cycle $v_1 - v_2 - \ldots - v_k - v_1$. Let the subtrees of $v_1, v_2, \ldots, v_k$ contain $x_1, x_2, \ldots, x_k$ elements. The idea is pretty much the same: we want to maximize the contribution of this cycle into the total distances. For that, we want to pair up the nodes in these subtrees, so that the total distance travelled along the edges of this cycle over all the paths between these pairs is as large as possible. How do we pair them up optimally?

Well, if we pair up nodes from subtrees of v_i and v_j, the shortest path between them will contain $min(|i - j|, k - |i - j|)$ edges of the cycle. Basically, we have the following problem now: we have n points on the circle: x_i at position i, and want to pair them up to maximize the sum of distances between them. How to do this?

Claim: it's optimal to pair opposite nodes. You can show this by simple exchange argument: if you paired nodes (a, b) and (c, d) with a, b, c, d going on circle in this order, you can change them to pairs $(a, c), (b, d)$, and the sum of distances won't decrease.

With this in mind, it's not hard to get the contribution of each cycle. Proving that it's achievable is left as an exercise for the reader.

Total asymptotic is then still just $O(n)$.