All-Ukrainian Collegiate Programming Contest 2022

II stage

Editorial

Thanks for participation!

,

Problem A. Adjacent Product Sum

Let $a_1 \geq a_2 \geq \ldots \geq a_n$. Then, we can show that one of the optimal ways to arrange them on the circle is: $a_1, a_3, \ldots, a_{2\lfloor \frac{n-1}{2} \rfloor + 1}, a_{2\lfloor \frac{n}{2} \rfloor}, \ldots, a_4, a_2$. In other words, first put all elements on odd positions, then all on even positions, in a different order.

For example, for $n = 7$, this would look as: $(a_1, a_3, a_5, a_7, a_6, a_4, a_2)$.

Let's show that it's optimal. We will use a simple fact: if $a \geq b$, $c \geq d$, then $ac + bd \geq ad + bc$ (as this is equivalent to $(a - b)(c - d) \geq 0$).

Let's show by induction that for each k there exists an optimal arrangement (optimal = with the largest possible sum of adjacent products) in which elements $a_1, a_2, \ldots, a_k$ form a consecutive segment, and numbers a_{k-1} and a_k are at the ends of this segment. To show this for $k = 2$, suppose that a_1 and a_2 aren't adjacent. Let the element directly clockwise to a_1 be a_x, and directly clockwise to a_2 be a_y. Then, let's reverse entire subsegment from a_x to a_2 (considered clockwise). The sum of products will change by $a_1a_2 + a_xa_y - a_1a_x - a_2a_y \geq 0$.

Now that we proved our statement for $k = 2$, let's make the induction step: consider an optimal arrangement in which this holds for $k - 1$. Wlog, in this segment a_{k-2} is the first element, and a_{k-1} the last clockwise (that is, if we traverse this segment clockwise, a_{k-2} is the first element we see, and a_{k-1} the last one.

Suppose that a_k isn't adjacent to a_{k-2}. Then, let a_x be the element next counterclockwise to a_{k-2} (by choice $x > k$), and a_y be the next element counterclockwise to a_k (by choice $x \geq k-1$). Then, let's reverse the (clockwise) segment from a_k to a_x. The sum will change by $a_ka_{k-2} + a_xa_y - a_{k-2}a_x - a_ka_y$, which is nonnegative as $a_{k-2} \geq a_y$ and $a_k \geq a_x$.

So, just sort the elements, arrange them in this order, and find the sum of products.

,

Problem B. Best Chess Piece

Let's find the smallest number of moves a prawn has to make to get from cell (a, b) to cell (c, d).

Let's also consider the chess coloring of the grid (that is, color cells (a, b) with $a + b$ even white and with $a + b$ odd black). Then, if prawn moves diagonally, the color of its cell doesn't change, and if it moves forward by one, the color of its cell changes.

First, prawn has to make at least $max(|a - c|, |b - d|)$ moves, as in one move prawn can only decrease $|a - c|$ and $|b - d|$ by at most 1. Second, if cells (a, b) and (c, d) have the same color, then this is always achievable. Indeed, let $max(|a - c|, |b - d|) = t$. At i-th move, just make a diagonal move so that for new cell (a_1, b_1) (to which prawn gets after this move) holds $max(|a_1 - c|, |b_1 - d|) \leq t - i$.

What if cells (a, b) and (c, d) have different colors? Then we would have to move forward at least once. As the order of moves doesn't matter, we might as well make it our first move. Then our goal would be to get from the cell $(a, b+1)$ to the cell (c, d) as fast as possible. But as we know, this takes $max(|a-c|, |b+1-d|)$ (as cells $(a, b + 1)$ and (c, d) have the same color).

So:

- If cells (a, b) and (c, d) have the same color, answer is $max(|a - c|, |b - d|)$
- Otherwise, it is $1 + max(|a - c|, |b + 1 - d|)$

Problem C. Cyclic Segment Sums

Let's try to see how arrays a with the same $f(a)$ look. Consider all arrays a for which $f(a) = b$. What do we know about them?

First, we know $a_{i+k} - a_i = b_{i+1} - b_i$. Consider elements $a_i, a_{i+k}, a_{i+2k}, \ldots, a_{i-k}, a_i$ (add k until we meet a_i again for the first time). What's the length of this cycle? It's $\frac{n}{\gcd(n,k)}$, where $\gcd(n,k)$ denotes the greatest common divisor of n and k.

Let's denote the set of indexes for which a_i is in this cycle by S. If at least for one $i \in S$, $b_{i+1} - b_i \neq 0$, then we can uniquely determine the values of all a_i for $i \in S$ (as $a_j - a_i$ can only be 1 when $a_j = 1, a_i = 0$, and can only be -1 when $a_j = 0, a_i = 1$). If all these b_i are equal, then all a_i are also equal, but here we have two choices: to make all of them equal to 0 or to 1.

Now, suppose that there are t cycles, in which all a_i have to be equal. If we choose x of these cycles to have all ones, and the rest to have all zeros, then the sum on each subsegment of length k increases by $x\frac{k}{\gcd(n,k)}$ (each cycle intersects with each segment of length k by exactly $\frac{k}{\gcd(n,k)}$ cells). So, for a given a, all arrays a_1 with $f(a_1) = f(a)$ look as follows:

- Consider all $\gcd(n,k)$ cycles $(a_i, a_{i+k}, a_{i+2k}, \ldots)$ for each i from 1 to $\gcd(n,k)$.
- For each of these cycles: if not all a_i in it are equal, then in all arrays a_1 with $f(a_1) = f(a)$, these a_{1i} will be the same as in a.
- If there are t cycles in which all a_i are equal, and x of them consist of all ones, then in a_1 in these cycles, all elements are also equal, and exactly x of them consist of all ones.

Now, let's count the number of different equivalent classes. Let $l = \gcd(n,k)$. Suppose that t of these cycles have all equal elements. Then there are $\binom{l}{t}$ ways to choose these cycles, and $2^{\frac{n}{l}} - 2$ ways to fill each other cycle. As for t cycles with equal elements, we only care how many of those cycles are all ones, and there are $t + 1$ possibilities. So, we have to find

$$\sum_{t=0}^{t=l}(t+1)\binom{l}{t}(2^{\frac{n}{l}} - 2)^{l-t}$$

Denote $2^{\frac{n}{l}} - 2 = a$. Note that $t\binom{l}{t} = \frac{l!}{(l-t)!(t-1)!} = l\binom{l-1}{t-1}$. Now, rewrite:

$$\sum_{t=0}^{t=l}(t+1)\binom{l}{t}a^{l-t} = \sum_{t=0}^{t=l}\binom{l}{t}a^{l-t} + l\sum_{t=1}^{t=l}\binom{l-1}{t-1}a^{l-t-1} = (a+1)^l + l(a+1)^{l-1}$$

(Here we just used that $(x+1)^k = \sum_{t=0}^{k} x^t\binom{k}{t}$)

So, we can solve each case in $O(\log)$ with simple binary exponentiation.

,

Problem D. Different XOR

Let X be the XOR of all the elements. If $X \neq 0$, we can split into two parts arbitrarily, and their XORs will be different.

If $X = 0$, then let's try to split the array into 3 parts and see when it's impossible. Let a_x be the first nonzero element of the array (if all elements are 0, clearly there is no solution). Let the first subarray be $a[1 : x]$. We want to split the rest into two parts so that XOR in both isn't 0 or a_x (the XORs of those parts can't be equal, as then the XOR of the entire array would be a_x, not 0).

So, if there is no such split, then for any y with $x + 1 \leq y \leq n$, XOR on subarray $a[x + 1 : y]$ is 0 or a_x. However, this means that each element there is 0 or a_x. So, all elements of the array are 0 or a_x. But in this case, XOR of any subarray is always 0 or a_x, so there can't be any such split.

So, the solution is: if $X \neq 0$, split arbitrarily. If $X = 0$, find the first nonzero element a_x, take subarray $a[1 : x]$, and try all ways to split the rest into two parts. If you haven't succeeded, there is no solution.

,

Problem E. Equalize

First, we can always make an array good in at most 2 operations. Indeed, first, append 0. The product becomes 0. Then, append the number equal to the minus sum of elements of the array, so that both product and sum are 0.

Now, we have to check if the answer can be 0 or 1. To check if it's 0, we have to check if $a_1 \cdot a_2 \cdot \ldots \cdot a_n = a_1 + a_2 \ldots + a_n$. Find the sum directly, and for the product, to avoid overflow, do the following: if it's above INF by absolute value, just set it to INF (where you can set INF to be, for example, 10^{12}, as sum won't ever exceed $10^6 n < 10^{12}$ by absolute value).

Now let's check if we can make an array good in 1 operation. Let the current sum be S, the current product be P. We have to check if there exists some integer x such that $S + x = Px \iff x(P-1) = S$.

Now, we have several cases.

- If $P = 1$, then such x exists if $S = 0$ and doesn't if $S \neq 0$
- If $P \neq 1$, then such x exists only if S is divisible by $P - 1$. If $S = 0$, it's always divisible. Otherwise, note that if $|P| \geq INF$, S can't be divisible by $P - 1$. So, in this case, check if P is not INF, and if so, check that S is divisible by $P - 1$.

Problem F. Form ABC

Clearly, there is no point in swapping adjacent equal characters. So, the relative order of As doesn't change, and the relative order of Cs doesn't change.

The condition that string s contains ABC as a subsequence can be rephrased as follows:

- Let L be the first position with $s_L =$ A, and R be the last position with $s_R =$ C. Then, check if $L < R$ and there is at least one B between positions L and R.

Then, let L be the first position with $s_L =$ A, and R be the last position with $s_R =$ C. Consider two cases.

- $L < R$. Then, if there is at least one B between s_L and s_R, the answer is 0. Otherwise, consider any X with $s_X =$ B. What's the smallest number of swaps needed to move this character between s_L and s_R?

  If $X < L$, then $L - X$. If $X > R$, then $X - R$.

  So, just find the minimum value of this over all X with $s_X =$ B.

- $L > R$. Consider any X with $s_X =$ B. What's the smallest number of swaps needed to put s_X between s_L and s_R?

  There are 3 cases:

  - $X < R \implies$ we need $L - X$ swaps.
  - $X > L \implies$ we need $X - R$ swaps.
  - $R < X < L \implies$ we need $L - R + 1$ swaps.

  Iterate through all such X, choose the smallest value.

Problem G. Graph and Hamiltonian Cycles

First, let's get some trivial bounds on the length of the shortest cycle. Take any position i ($1 \le i \le 2n-1$). Let the numbers of `W`, `B` in $s[1:i]$ be $left_W, left_B$ correspondingly.

Consider any Hamiltonian cycle. Suppose that it contains x edges connecting node from $\{1, 2, \ldots, i\}$ to $\{i+1, i+2, \ldots, 2n\}$. The degree of each node in Hamiltonian cycle is 2, so the number of edges connecting nodes from $\{1, 2, \ldots, i\}$ is $\frac{2i-x}{2}$. So, x has to be even. Also, note that the number of edges connecting nodes from $\{1, 2, \ldots, i\}$ can't exceed $2min(left_W, left_B)$. So,

$$\frac{2i-x}{2} \le 2min(left_W, left_B) \implies x \ge 2(left_W + left_B) - 4min(left_W, left_B) = 2|left_W - left_B|$$

Also, of course, $x \neq 0$, as otherwise graph wouldn't be connected. So, $x \ge 2max(1, |left_W - left_B|)$.

It turns out that these bounds are all achievable, so if we just wanted to learn the length of the shortest, we could just find all these bounds over each position i and sum them up. (Proof that these bounds are achievable will follow from what's written below).

But we are interested in the number of such cycles. For that, let's analyze the structure of cycles in which all of these bounds hold. Again, choose some i and define $left_W, left_B$ as before.

It's easy to see that:

- If $left_W > left_B$, then the graph on $\{1, 2, \ldots, i\}$ consists of $left_W - left_B$ paths of form $WBWBW \ldots BW$.
- If $left_B > left_W$, then the graph on $\{1, 2, \ldots, i\}$ consists of $left_B - left_W$ paths of form $BWBWB \ldots WB$.
- If $left_B = left_W$, then the graph on $\{1, 2, \ldots, i\}$ consists of a single path $WBWB \ldots WB$.

It's easy to see that if this holds for every i, then all bounds on the numbers of edges which connect a node from $\{1, 2, \ldots, i\}$ to $\{i+1, i+2, \ldots, 2n\}$ will match exactly.

Now, let dp_i denote the number of ways to connect edges on the first i nodes so that all first i conditions hold. How do we update this dp?

Assume now $left_W > left_B + 1$. If the next character is `W`, then we have to add a path from a single node. If the next character is `B`, then we have to reduce the number of paths by 1, by connecting `B` to the ends of some two paths. But here we run into a problem: it matters if the path has a length larger than 1 or not, as in that case, it has two "distinct"ends or not. To solve this issue, let's modify our problem a bit.

Let's find the number of oriented Hamiltonian cycles. Then we could divide this number by 2 and obtain the answer to the initial problem. All the conditions remain the same, just paths become oriented.

Now, making a transition is easy! As now, when we are processing `B` in the case above, we don't have to choose 2 ends; we have to choose a right end and a left end. There are precisely $(left_W - left_B) \cdot (left_W - left_B - 1)$ to do so.

Now updating the dp is easy; you are just dealing with some cases. Briefly:

- $left_W > left_B$. If next character is `W`, $dp_{i+1} = dp_i$. If next character is `B`, then, if $left_W > left_B + 1$, $dp_{i+1} = dp_i(left_W - left_B) \cdot (left_W - left_B - 1)$. If $left_W = left_B + 1$, $dp_{i+1} = 2dp_i$ (there are two ways to prolong the chain).
- $left_B > left_W$ is symmetric.
- $left_W = left_B$. Wlog next character is `W`. Then we have to connect this `W` to the only black end of the previous chain, so $dp_{i+1} = dp_i$.

We can calculate this in $O(n)$.

Problem H. Hating 3-Colorability

We will show that the answer is always 2 or 3. Let's initially color nodes of the first part (from 1 to n) in color 1 and the remaining in color 2.

It's clear that if we don't add any edges, the graph remains 2-colorable (and therefore 3-colorable).

If we add one edge (u, v), then just color node v in color 3, coloring will remain proper.

Now, suppose that we added two edges, and the graph is no longer 3-colorable. Let's consider some cases. Suppose that one of these two edges connects two nodes from different parts. Then, if the other edge is (u, v), we can just color node v in color 3, and the coloring will remain proper. So, we have to add edges, which connect nodes from the same part.

Let these edges be (u_1, v_1) and (u_2, v_2). If these edges share a node, say, $u_1 = u_2$, then we can just color node u_1 in color 3, and the coloring will be proper. Otherwise, let's try coloring some endpoint of each of these two edges in color 3. If we can't get proper coloring this way, it means that all of the edges $(u_1, u_2), (u_1, v_2), (v_1, u_2), (v_1, v_2)$ are present in this graph. And indeed, if such a cycle of length 4 is present in the graph, we can add edges (u_1, v_1) and (u_2, v_2), obtaining a subgraph on 4 nodes u_1, v_1, u_2, v_2, in which each pair of nodes is connected, and which, therefore, isn't 3-colorable.

Even if there is no 4-cycle, we can always add at most 3 edges to make the graph not 3-colorable. It's enough to show that we can choose 2 nodes u_1, v_1 from the first part, and nodes u_2, v_2 from the second part, such that at least 3 edges among edges $(u_1, u_2), (u_1, v_2), (v_1, u_2), (v_1, v_2)$ are present in this graph (then we can add all remaining edges to obtain a complete graph on 4 nodes). Suppose that it doesn't exist. Then there is no edge (u, v) with u in the first part, v in the second, such that there is at least one more edge from u and at least one more edge from v. Then for any edge (u, v), at least one of u, v is a leaf. Then consider any non-leaf node u, it can be connected only to leaves, so u together with these leaves forms a separate connected component, and the graph is not connected. This contradicts the problem statement (graph is told to be connected).

So, answer is 2 if there is a 4-cycle u_1, v_2, v_1, u_2, and 3 otherwise. How to check if there is a 4-cycle?

Here is an algorithm that, for a given node, checks if it's in some 4-cycle in $O(n)$, giving total runtime of $O(n^2)$. Consider node v, and it's neighbors $u_1, u_2, \ldots, u_k$. v is contained in a 4-cycle iff there is a node $v_1 \neq v$, which is connected to at least two nodes among $u_1, u_2, \ldots, u_k$.

Then, just start going through all neighbors of nodes $u_1, u_2, \ldots, u_k$, marking nodes which we meet (except v). If we have to mark some node twice, we found a 4-cycle. This takes $O(n)$ per node.

,

Problem I. Increasing Split Swapper

Let's call positions pos for which $p_{pos} < pos$ **small**, and positions with $p_{pos} \geq pos$ **big**.

We will show the following statement:

Lemma: Permutation can be split into two increasing subsequences **iff** elements on small positions form an increasing subsequence, and elements on big positions form an increasing subsequence.

Proof: If both these subsequences are increasing, we found the required split. Let's show that if a permutation is splittable, then these subsequences are increasing.

First, note that splittable permutation can't contain any decreasing subsequence of length 3, as no two of such 3 elements can be in the same increasing subsequence.

Now, suppose that the sequence on elements on small positions is not increasing. Then, there exist some x, y with $x < y, p_x < x, p_y < y, p_x > p_y$. Consider first x elements, there has to be some element $\geq x$ there. This element, together with p_x and p_y forms a decreasing subsequence of length 3.

Now, suppose that the sequence on elements on small positions is not increasing. Then, there exist some x, y with $x < y, p_x \geq x, p_y \geq y, p_x > p_y$. Consider last $n - y + 1$ elements. $p_x > p_y \geq y$, so there are at most $n - y$ elements which are $\geq y$ there. Then p_x, p_y, and the last element smaller than y form a decreasing subsequence of length 3.

Now that the lemma is proved, to solve the problem, we just have to be able to support these sequences of elements on small/big positions and to answer if they are increasing.

Let's keep a set of all small positions and a set of all large positions. For each set, we will keep the number of such x, that $p_x > p_{nxt(x)}$, where $nxt(x)$ is the smallest element of that set larger than x. The swap implies $O(1)$ inserts/deletions, and each insertion/deletion takes $O(\log)$ to process.

Total complexity $O((n + q) \log n)$.

,

Problem J. Joke

Let x be the largest value in the array a, then x satisfies all conditions from the statement. It indeed appears in a, and for any i, j, $1 - x \le a_i - a_j \le x - 1$, so $|a_i - a_j| \le x - 1$.

,

Problem K. Knowledgeable Andrii

Let's add an edge between each i, j, the distance between which is odd. Let's show that if there is any graph for which the distances match, this graph also has to work.

For any i, j between which the distance is odd, the parity will match. For any i, j between which the distance is even, say, $2t$, there is a node k on the shortest path between them, for which $d(k, i) = 1$, $d(k, j) = 2t - 1$. Therefore, we drew edges (k, i) and (k, j), and the distance between i, j in our graph is 2, so the parities also match.

So, we should just add all these edges, and check if the graph is connected and all parities match. $O(n^3)$ with Floyd-Warshall.

Problem L. Least Annoying Constructive Problem

Odd n. Let $n = 2k + 1$.

We will imagine these nodes on a circle, numbered from 1 to n, and use the cyclic notation of nodes; that is, node $n + i$ is the same as node i for any i.

Consider the following construction. For each i from 1 to n, output the following k edges: $(i, i + 1), (i - 1, i + 2), (i - 2, i + 3), \ldots, (i - k + 1, i + k)$. On the circle, view these as k edges parallel to edge $(i, i + 1)$. Let's call such k edges for a given i block i.

Let's show that this works. Consider any $n - 1 = 2k$ consecutive edges. Such $2k$ edges completely contain some block, wlog block 2, and contain some suffix of block 1, let's say of length x, and some prefix of block 3 of length $k - x$.

After we drew block 2, we have $k + 1$ components remaining: k edges and a single node $3 + k$. The idea is that the i-th edge of block 1 connects i-th and $i + 1$-st of these components, the same for i-th edge of block 3. Then it's easy to see that the suffix of block 1 of length x connects the last $x + 1$ components into a single component, and the prefix of block 3 of length $k - x$ connects this resulting component and the first $k - x$ components.

Even n. Let $n = 2k + 2$. Let's once again draw $2k + 1$ nodes on a circle, and let the center of this circle be point $2k + 2$. We will once again number the points on the circle cyclically mod $2k + 1$ (note that this doesn't apply to the center node $2k + 2$).

Then, let the "block i"consist of the following k edges for each i from 1 to $2k + 1$: $(i, i + 1), (i - 1, i + 2), (i - 2, i + 3), \ldots, (i - k + 1, i + k)$, **and** the edge $(i + k + 1, 2k + 2)$. You can visualize this as: draw k edges parallel to $(i, i + 1)$, and the edge from the center to the last remaining point. Then, write down edges of these blocks for each i from 1 to $2k + 1$, in this order.

The proof that this works is the same as in the odd case.