Problem A. Reversing

Author: Sunghyeon Jo (ainta)

Observe that, a cell's initial state should be the same as the current state if there is an adjacent cell that has different color. If all adjacent cells are the same color with current cell, then it could be either white or black at the initial state. Therefore, if there are total C cells that all adjacent cells have the same color with themselves, then the answer is 2^C.

Problem B. Lawyers

Author: Sunghyeon Jo (ainta)

Make a digraph $G = (V, E)$ such that $(A, B) \in E$ if and only if lawyer A can defend lawyer B. The goal is determining whether it is possible to choose edges such that every vertex has positive indegree and no two edges with the same two endpoints and opposite direction are selected.
If $(u, v) \in E, (v, u) \notin E$ satisfies, then it is always advantageous to select (u, v). Make it selected and remove all such edges (u, v) from G. Mark vertex v as it has a positive degree.
After removing edges, all edges are pairs of opposite edges. Consider edges as bidirectional.
If a connected component has at least one marked vertex u, then there is a spanning tree T. It could be seen as a directional rooted tree with u and selecting edges in the tree guarantees that every vertex of the component has positive indegree.
Also, if a connected component with k vertices has at least k edges, then there will be at least one edge not included in a spanning tree T. Select the edge of any direction will make a marked vertex, so it can be resolved as former case.
The only remaining case is there is a connected component with k vertices and $k-1$ edges and has no marked vertex. Since only $k-1$ edges can be selected, it is not possible to select edges to satisfy the condition.

Problem C. One, Two, Three

Author: Sunghyeon Jo (ainta), Jeyeon Si (tlwpdus)

First, let's limit the form of the answer. If we consider a candidate answer where (1 2 3) and (3 2 1) are well-arranged so that they don't overlap, it is nothing worse to push 1 in (1 2 3) as far to the left as possible and 3 as far as possible to the right. Similarly, (3 2 1) would be nice to push the 3 as far to the left and the 1 as far to the right.
Let A be the number of matches (1 2 3), B be the number of matches (3 2 1). Then it is sufficient to consider only answers in which the first A one and the last A threes are matched together and the first B threes and the last B ones are matched together.
Think of two (1 2 3) matching x and y. If the order of ones in x and y do not agree with order of twos in x and y (i.e. 1 in x is left from 1 in y but 2 in x is right from 2 in y), it is possible to swap two 1s in x and y. Likewise, if order of threes do not agree with order of twos, it is possible to swap two 3s. By such swapping

process, we can obtain an **ordered** matching: for any two (1 2 3) matching x and y, the relational order of position of same number in x and y are same for 1, 2, and 3.
Accumulating the above observations, We can construct an algorithm that determines if it is possible to make A of (1 2 3) matchings and B of (3 2 1) matching:

1. If there is less than $a+b$ ones, it is impossible to make it. The same is applied to two and three, too.
2. Since there is at least $a+b$ of each kind of numbers, the first a ones and last b threes are disjoint with the last b ones and the first b threes. By observation of **ordered** matching, we know which ones and threes should be in the same matching.
3. The remaining problem is to assign 2 for each matched pair of 1 and 3. 2 should be located between 1 and 3. It is just a matching problem between points and segments (ranges between 1 and 3), so it is can be solved in linear time after sorting.

The algorithm above gives $O(N^3)$ solution for the original problem, and it can be reduced into $O(N^2 \log N)$ and $O(N^2)$ using the monotone property of possible (A, B) pairs. But there should be more observation to solve it in linear or near-linear time.

Assuming A and B are given. As suggested in the algorithm, it can be expressed as bipartite matching of twos and ranges between (1 3) or (3 1) pairs. See ranges as vertices on the left(L) and each 2 as vertices on the right (R) in the bipartite graph. There is an edge between $u \in L$ and $v \in R$ if and only if 2 of v is included in range u. We want to know the existence of a matching of size $A + B = |L|$.

As **Hall's marriage theorem** states, matching of size $|L|$ exists if and only if when $|N(S)| \geq S$ holds for every $S \subset L$. It means that for any k ranges in L, there is at least k twos that included in at least one of the k ranges.
Let's define some notations:

- Define $F(l, r)$ as number of ranges in L which included in range $[l, r)$.
- Define 1_i as $|\{j|A_j = 1, j < i\}|$, 2_i as $|\{j|A_j = 2, j < i\}|$, 3_i as $|\{j|A_j = 3, j < i\}|$

And we can see the following observations:

- Matching of size $|L|$ exists if and only if $F(l, r) \leq 2_r - 2_l (0 \leq l \leq r \leq N)$

"only if" is induced directly from Hall's theorem. "if" holds since we can focus only on contiguous ranges.

How to calculate $F(l, r)$ efficiently? Let's considering only (1 3) ranges. as A increases, $F(l, r)$ increases like $0, 0, .., 0, 1, 2, 3, 4, ...$ Precisely, from A = (number of ones left to l) + (number of threes right to r), $F(l, r)$ increases by 1. It can be formulated as $\max(A - 1_l - (3_N - 3_r), 0)$. Likewise, number of (3 1) ranges included in $[l, r)$ is $\max(B - 3_l - (1_N - 1_r), 0)$.
Therefore, $F(l, r) = \max(A - 1_l - (3_N - 3_r), 0) + \max(B - 3_l - (1_N - 1_r), 0)$
So, we want to find (A, B) with maximum $A + B$ such that
$F(l, r) = \max(A - 1_l - (3_N - 3_r), 0) + \max(B - 3_l - (1_N - 1_r), 0) \leq 2_r - 2_l$ holds.
This can be divided into four inequalities:

- $A - 1_l - (3_N - 3_r) + B - 3_l - (1_N - 1_r) \le 2_r - 2_l$
- $A - 1_l - (3_N - 3_r) \le 2_r - 2_l$
- $B - 3_l - (1_N - 1_r) \le 2_r - 2_l$
- $0 \le 2_r - 2_l$

The last one is vaguously true, and first three can be restated as:

- $A + B \le min_{0 \le l \le r \le N}(1_l + 3_N - 3_r + 3_l + 1_N - 1_r + 2_r - 2_l)$
- $A \le min_{0 \le l \le r \le N}(1_l + 3_N - 3_r + 2_r - 2_l)$
- $B \le min_{0 \le l \le r \le N}(3_l + 1_N - 1_r + 2_r - 2_l)$

The right-hand side of each inequality is can be calculated in $O(N)$ by calculating minimum/maximum of prefix/suffix. Among the (A, B) satisfying those inequalities, we can find the one with maximum $A + B$ easily.
After finding (A, B), we can construct the answer as we stated at first.
Overall, we can solve this problem in linear time.

Problem D. Lonely King

Author: Sunghyeon Jo (ainta)

First of all, when a blue path is changed into a red edge, there is no case that it causes more contact than before. The initial state is a directed rooted tree. If there is an edge into a non-leaf vertex v in the final states, there must be an edge from v too, so it could be merged to an edge. Therefore, there is an optimal final state in which every edge's endpoint is a leaf vertex of the initial tree.

With this in mind, the problem is to divide the edges of the directed rooted tree into paths (u, v) where v should be a leaf of the initial tree, and the goal is to minimize the $\sum C_u C_v$.
If there is a red edge (u, v) is a final state, let's call v connected to u. Consider a vertex u which is not the root. Let's call subtree with root u T_u. Then there is exactly one leaf node in T_u which is connected to u's ancestors.
From this, we can think a dynamic programming solution like below:

- $D[u][l]$: minimum cost could be achived when l is the leaf connected to u's ancestors.
- $D[u][l] = \min(D[x][l] + \sum(D[c][l_c] + C_u C_{l_c}))$
  where x is u's child node which is ancestor of l and c is all other childs of x. l_c is a leaf node minimizes $D[c][l_c] + C_u C_{l_c}$.

For every leaf l in the subtree T_u, consider a line of $y = C_l \cdot x + D[u][l]$. We can see that only the lower hull of the lines are used in above DP formula. If we have lower hull of those lines for each T_c for u's child c, then we can calculate lower hull for T_u.
We can see that lines consisting lower hull of T_u is union of lines of T_c where c is child of u, shifted along the direction of y-axis.

Since merging two lower hull of size A and B can be done in $O(min(A,B)\log min(A,B))$ time with a data structure like set of balanced BST, the whole process can be done in $O(Nlog^2N)$ time (small-to-large trick).

Problem E. Treasure Box

Author: Sunghyeon Jo (ainta)

First, if the initial state is a palindrome, the answer is 0. Otherwise, let given string as $S[1\ N]$. Let b be the smallest i where $S[i]$ is different from $S[N+1-i]$,and $e = N-b+1$.
we know that to make a given string a palindrome, we need to swap either b-th character or e-th character. Also, since there is no reason to replace the characters outside the $[b,e]$ range, if $H[i]$ is the HP to be consumed when the starting position is i,

- If $i < b$ then $H[i] = H[b] + C(b-i)$
- If $i > e$ then $H[i] = H[e] + C(i-e)$

holds.
So we only need to solve the problem when the starting point is in $[b,e]$. Considering the case of replacing b-th character among b-th and e-th, the possible paths when the starting position is the x-th character can be classified into the following two types.

- $x \to r \to b$ (if you go from x to the right and then to the left to get to b)
- $x \to b \to r$ (if you move left from x to b and then right)

In both cases, it needs to be possible to make a string into a palindrome by replacing only the characters of $[b,r]$, and if $[b,r]$ includes both corresponding characters in the palindrome, choosing the one which consumes less HP is always an advantage.
We can pre-store the array $P[r]$, the HP required to replace the characters to make a palindrome when considering interval $[b,r]$. When x is given, $min_r P[r] + C(r-x) + C(r-b)$ is the optimal value of the path $x \to r \to b$. Likewise, $min_r P[r] + C(x-b) + C(r-b)$ is the optimal value of the path $x \to b \to r$. And it can be computed in $O(N)$ time for all x if x is moving by one space to the left starting from e.
The case replacing e-th character instead of b-th can calculated in the same way if we think symmetrically. Therefore, we can solve this problem in $O(N)$ time.

Problem F. Beautiful Sequence

Author: Sunghyeon Jo (ainta)

A sequence of N integers, $A_1, ..., A_N$ is given. The goal is to rearrange the sequence to maximize the number of elements which is not less than adjacent elements.

For $1 \le k \le N$, call k a **beautiful** position if it satisfies $A_{k-1} \le A_k \ge A_{k+1}$ (Assume $A_0 = A_{N+1} = -INF$). Otherwise, call k a awful position.

Let awful positions be $i_1, i_2, \cdots, i_m$ and $i_0 = 0, i_{m+1} = N+1$. Since $i_k+1, ..., i_{k+1}-1$ are beautiful position, $A_{i_k} \le A_{i_k+1} = A_{i_k+2} = \cdots = A_{i_{k+1}-1} \ge A_{i_{k+1}}$ must be satisfied.

To maximize the beauty of the sequence, m, number of the awful positions should be minimized.

Sequence with at most m awful positions can be generated by rearranging if it is possible to choose $V_1, \cdots, V_m$ as:

1. Among the N elements of $A_1, \cdots, A_N$, choose m numbers $V_1 \le V_2 \le \cdots \le V_m$. They will be numbers that will be placed in a awful position.
2. Let U as a sorted set of not selected elements without duplication. $U = \{U_1, .., U_k\}, U_i < U_{i+1}$.
3. $V_i \le U_i (1 \le i \le \min(m, k))$ must be satisfied.
4. $k \le m + 1$ must be satisfied.

Alternating sequence of $U_1, V_1, U_2, V_2, \cdots$ is a beautiful sequence with possibly m awful positions since each U_i is not smaller than V_i and V_{i-1}. Replacing each U_i to its copies (to make this sequence $\{A_i\}$'s permutation) do not increase awful positions. Therefore, there is a rearrangement of $\{A_i\}$ which has at most m awful positions.

Conversely, if it is possible to rearrange given sequence into a sequence with at most m awful positions, then it is possible to choose $\{V_i\}$ as suggested above. It is could be done by assigning $\{V_i\}$ as all numbers in awful positions.
For example, when $A = [2, 1, 3, 1, 4, 1, 5, 1, 6]$, by choosing $V = [1, 2, 3]$, we can obtain $A' = 1, 1, 1, \mathbf{1}, 6, \mathbf{2}, 5, \mathbf{3}, 4$ with 3 awful positions.

Then how to find minimum m such $V_1, \cdots, V_m$ exists?
Let A's distinct elements as $a_1 < \cdots < a_k$ and c_i be the occurrence of a_i in A.
Starting from empty sequence and empty V and W(W is the multiset of elements not in V), we will add c_i copies of a_i to the sequence from $i = 1$ to $i = k$ and add elements to V if needed.
Assume that after adding c_i copies of a_i, the condition $|V| + 1 <$ (number of different kinds of numbers in W) holds. Then we need to select an element in W and move it to V.
In order to minimize the addition of elements to V, (number of different kinds of numbers in W) should be minimized. Thus, it is optimal to choose the number with least occurrence in W and add it to V when V needs more elements.
After executing above process from $i = 1$ to $i = k$, $|V|$ is the minimum possible value of number of awful positions and the maximum beauty can be calculated directly.
The whole process above can be done in $O(N \log N)$ time by using data structures like priority queue. Therefore, the whole problem is can be solved in $O(N \log N)$ Time.

Problem G. Make Everything White

Author: Sunghyeon Jo (ainta)

It is enough to use only operation 2 and operation 3. Consider state of the grid M after executing operation 2 for all cells. If a cell's color is black in M, then changing its color to white without changing other cells could be done by replacing the operation 2 on the cell by operation 3. Therefore, after doing operation 2 on white cells in M and doing opration 3 in black cells in M, we could obtain a grid with every cell colored white.

Problem H. Optimal Quadratic Function

Author: Sunghyeon Jo (ainta)

For two functions f, g which has error $< K$, $\frac{f+g}{2}$ has error $< K$. Therefore, we can use ternary search for a and b, the coefficient of the quadratic term and linear term.
Let the absolute or relative error as ε and the range of coordinates as L. Then we can solve this problem with time complexity $O(N \log^2(L/\varepsilon))$.
However, it is not easy to handle floating point precision in this problem. If we naively use the formula $f(x) = ax^2 + bx + c$, the c could be massive, like $O(L^2)$ scale. Therefore, it is not easy to avoid wrong answer in this way.
Among the given N points, let the maximum value of x be x_r and minimum value of x be x_l. Let $f(x) = a(x-l)(x-r) + b(x-l) + c$.
When f is the optimal quadratic function, following holds for $l \le x \le r$:

- $max_i|f(x_i) - y_i| \le max_i(|y_i - 0|) \le L$
- $|c| = |f(l)| \le L + |y_i| \le 2L$
- $|b(x-l)| \le |b(r-l)| = |f(r) - c| \le L + |y - c| \le 4L$
- $|a(x-l)(x-r)| \le |f(x) - b(x-l) + c| \le y + |f(x)| + |b(x-l)| + |c| \le 8L$

Therefore, if we use above formula for f and do ternary search, the scale of whole terms are $O(L)$ scale so it is possible to avoid big precision error. And we can prove that it is enough to use $f(x) = a(x-l)^2 + b(x-l) + c$, too. So that we can solve the problem after shifting each the x_i by $-x_1$, and just use ternary search for $f(x) = ax^2 + bx + c$.

The time complexity $O(N \log^2(L/\varepsilon))$ can be reduced into $O(N \log(L/\varepsilon))$.
Use ternary search for a, so assume that a is fixed.
Then our goal is decide b, c to minimize maximum of $|f(x_i) - y_i| = |a(x_i)^2 - y_i + bx_i + c|$.
And if b fixed, minimized maximum of $|f(x_i) - y_i|$ is $0.5\times$ (difference between min and max of $bx_i + a(x_i)^2 - y_i$)
Let $z_i = a(x_i)^2 - y_i$. we want to choose b that minimizes the difference between min and max of $bx_i + z_i$.
And the min and max of $bx_i + z_i$ are from convex hull of (x_i, z_i). If we use rotating calipers algorithm on this, finding optimal b can be done in $O(N)$ time.
Therefore, we can solve whole problem in $O(N \log(L/\varepsilon))$ time due to ternary search for a.

Note: For $O(N \log(L/\varepsilon))$ algorithm, the precision error could be bigger than slower one, so long double should be used to manage it.

Challenge: there seems linear solution using linear programming: link

Problem I. Visiting Friend

Author: Sunghyeon Jo (ainta)

Build a block-cut tree of the graph $T = (V, E)$, where V is union of original vertices and biconnected components and E is set of (v, b) where v is vertex included in the biconnected component b.

Assume that A and B are not in ancestor-descendant relationship in the block-cut tree. Then, the answer is N - (number of original vertices in subtree of A) - (number of original vertices in subtree of B) since we can't visit vertex B from the verticies in the subtree of A without visiting A, and vice versa.

It can be handled similarly even if A is B's ancestor or converse case.

Problem J. Cooperation Game

Author: Sunghyeon Jo (ainta)

If there are students from same class with indices $a_1 < a_2 < \cdots < a_k$, then it is best to come out in order of $(a_1, a_k), (a_2, a_{k-1}), \cdots$. From this we can make pairs of students and decide orders of pairs from the same class.
For four indices $a < b < c < d$, we can get the same score regardless of order of pair (a, c) and (b, d). Also, if there is (a, d) and (b, c) then (a, d) should be executed before (b, c).
Therefore, it is optimal to do them in order of the largest difference between the indices of the two students forming a pair.

Problem K. Connect the Dots

Author: Sunghyeon Jo (ainta)

We can bend the Ox axis to a circle and consider the problem as about the connecting points on the circle. The condition of two different curves cannot have a common interior point is equivalent to two chords of the circle cannot have a common interior point.

First, consider only chords between adjacent points. If two adjacent points have different colors, then it is always better to connect the two.
Then we can think of chords between non-adjacent points. since it can be considered as a diagonal of N-gon, there could be at most $N - 3$ such chords.

Lemma 1. *If there is at least 3 different colors, it is always possible to draw $N - 3$ chords satisfying all conditions.*

Proof. Group points as blocks, a contiguous points with same colors. If there is at least 3 different colors, 3 adjacent blocks A, B, C with different color always exists. If we read color of the points in those blocks, it is like "AA...ABB...BBC..CC". Draw chords between last A and second, third, ... , last B, and the first C. If the number of points of block B is n_B, then after drawing chords, we can see the new state as block B has removed (so n_B vertices removed) and n_B new chords added.
Also, think of draw chords between last A and second, third, ... , last B, but not first C. then after drawing chords, we can see the new state as only one point of block B remaining (so $n_B - 1$ vertices removed) and $n_B - 1$ new chords added.
We can think above two operations as AA..AABB..BBCC..CC $\Rightarrow$ AA..AACC..CC and AA..AABB..BBCC..CC $\Rightarrow$ AA..AABCC..CC.
By executing these two operations, it is possible to leave only one points for each color. Then drawing all chords from one vertex is enough. $\square$

Construction by above process can be done in $O(N)$ by data structures like linked list.

Since there will be no chords when there is only one color exists, the only case remaining is when only two color has used.
When only 2 colors were used, then the graph is a bipartite graph so there is no odd cycle and the possible shortest length of the cycle is 4. Let the number of the blocks C. Then, the upper bound of number of the chords (including chords between adjacent points) is $N - 2 + C/2$. This is because if the colors of the points are alternating, then $C = N$ and the maximum number of chords is $3N/2 - 2$ by Euler's formula. And it is not difficult to find way to construct $N - 2 + C/2$ chords.
Time complexity: $O(N)$