Problem A. Yo-Gi-Oh!

Finding vertex b is equivalent to finding vertex a on the reversed di-graph.

Assume we start a DFS from some arbitrary vertex of the given graph. If the DFS ends up visiting all vertices, then we found our answer. If it doesn't, then we know our answer, if it exists, is one of the remaining unvisited vertices. We thus relaunch the DFS from another arbitrary unvisited vertex until none remain.

It can be proven that the vertex a, if it exists, is the last vertex we launch a DFS from.

Problem B. Youtube Channels

Brute force solution

Keep track of an array with the number of followers for each channel.

- Queries of type 1 and 2: Updates in O(1) by updating the number of followers in the array.
- Queries of type 3 and 4: Looping through the array O(n) and keep the index and the number of the followers corresponding the the channel with min (or max) followers.

Overall complexity: $O(q \times n)$

Using Segment Tree

Keep track of the number of the followers for each channel. Build two segment trees, one for the min and one for the max. Each node in the segment tree contains the index of the channel with the min (or max) number of followers, in the sub-array it represents.

- Queries of type 1 and 2: Update the value of a node by comparing the number of followers of the channels in the sub left and right parts.
- Queries of type 3 and 4: The channel with the min (or max) number of followers is always at the root of the corresponding segment tree.

Overall complexity: $O(q \times log(n))$

Problem C. Find a Name

The naive solution of this problem iterates over all possible permutations of the array, and checks each one for the given properties. The complexity of such solution is $O(N * N!)$, which is obviously too slow to pass!

The first observation to make is the actual values of the array do not matter since they're distinct, and thus can be remapped to a permutation of natural numbers from 1 to N. Moreover, the values left to use when building a partial solution do not matter either! What matters is the relative differences between elements only.

Assume we have a recurrence F defined as follows:

- $F(left, right, dir, streak)$ is the number of permutations starting from an element with $left$ values remaining that are less than the previously placed value, $right$ values remaining that are greater than the previously placed value, values can only be placed in the increasing/decreasing direction depending on the value of the boolean dir, and the increasing/decreasing trend has continuously been going for $streak$ consecutive values before the placement of the current one.

- $F(0, 0, dir, streak) = 1$
- $F(left, right, dir, streak) = 0$ for any $streak \geq k$
- If we are currently going in the increasing dir, we try all elements in $right$
- If we are currently going in the decreasing dir, we try all elements in $left$

Using dynamic programming, the answer to the problem $F(left_{initial}, right_{initial}, 0, 0)+F(left_{initial}, right_{initial}, 1, 0)$ can be computed in $O(N^4)$

Problem D. Chaimae and Cellular Towers

This problem is a classical geometry problem where we have to find the area of the non-intersecting region between two circles.

The first step that should be done is to calculate the Euclidean distance between the centers of the two circles by the following formula: $d = \sqrt{(x_d - x_c)^2 + (y_d - y_c)^2}$. Based on d, we have four cases to be considered:

- First case: $d = 0$ and $r_c = r_d$: the answer is $A = 0$ since it's the same circle.
- Second case: $d < |r_c - r_d|$ which means a circle is inside another, so the answer is $A = |A_c - A_d|$ where A_c and A_d are the area of circle C and the area of circle D respectively.
- Third case: $d < r_c + r_d$ which means that they do not intersect, so the answer is: $A = A_c + A_d$.
- Fourth case: When the two circles partially intersect (when all of the previous cases are not satisfied). The first step is to calculate the area of the intersection. There are different solutions to this problem. The idea that we will explain in this editorial is based on calculating the areas of the circle sectors and subtracting the areas of the triangles, so we will get the area of the lenses.

Circles' Equations

We have the equations of the two circles: $x^2 + y^2 = {r_c}^2$ and $(d - x)^2 + y^2 = {r_d}^d$ for circles C and D respectively.

Solving for x: $x = \frac{{r_c}^2 - {r_d}^2 + d^2}{2d}$. Using the Pythagorean equation, we solve for y: $y = \sqrt{{r_c}^2 - x^2}$.

Area of Circle Sector

The angles θ_c and θ_d are the angles of circle sector for both circle C and D respectively.

$\theta_c = 2sin^{-1}(\frac{y}{r_c})$ and $\theta_d = 2sin^{-1}(\frac{y}{r_d})$

Area of sector of circle C: $A_{sc} = \frac{\theta_c}{2\pi}\pi{r_c}^2 = {r_c}^2 sin^{-1}(\frac{y}{r_c})$

Area of sector of circle D: $A_{sd} = \frac{\theta_d}{2\pi}\pi{r_d}^2 = {r_d}^2 sin^{-1}(\frac{y}{r_d})$

Area of Triangles

Area of triangle of circle C: $A_{tc} = \frac{1}{2}{r_c}^2 sin(\theta_c)$

Area of triangle of circle D: $A_{td} = \frac{1}{2}{r_d}^2 sin(\theta_d)$

Area of Intersection

Area of intersection is the sum of the areas of circles' sectors minus the areas of the triangles: $A_{intersection} = A_{sc} + A_{sc} - A_{tc} - A_{rd}$.

$A_{intersection} = {r_c}^2 sin^{-1}(\frac{y}{r_c}) + {r_d}^2 sin^{-1}(\frac{y}{r_d}) - \frac{1}{2}{r_c}^2 sin(\theta_c) - \frac{1}{2}{r_d}^2 sin(\theta_d)$

So, the final solution is: $A = A_c + A_d - 2A_{intersection}$ where A_c and A_d are the areas of circles C and D respectively. Please note that the area of intersection must be multiplied by 2 because it was calculated twice by calculating the areas of the two circles.

Time Complexity: $O(1)$

Problem E. Wissal's Workout

Quadratic time per query

For every query, naively check the correctness of every cyclic shift of the subarray. Since every subarray has $O(n)$ cyclic shifts, and checking each shift takes linear time, the complexity of this solution is $O(q \times n \times n)$

Linear time per query

We need to be able to count the number of correct cyclic shifts in linear time. Let's build the partial sum array of the initial array. This would show us the relative increase or decrease between consecutive elements of the subarray.

We must observe that the partial sums of the cyclic shifts of a subarray all have the same shape, and differ by only a constant positive or negative shift. Thus, they all have the same number of absolute minima.

The number of absolute minima of the partial sums of any subarray with 0 net sum is the answer to the query. These can be found in linear time. The complexity of this solution is $O(q \times n)$

Logarithmic time per query

Use a segment tree to count the minimas for each query.

Overall complexity: $O(q \times log(n))$

Problem G. Busy Drivers

A key observation to solve this problem is to visualize it as a graph problem.

We can create a Graph with N nodes each representing a travel request, and there is an edge between two nodes in this graph if and only if request a_i can be processed after request a_{i-1}.

Now what we can observe is the answer is the number of vertex disjoint paths in this graph, and we can see that this can be solved by matching each vertex with some ancestor let's call it u, and u will be matched to only the current vertex, or that request is the itself the start of a new path.

This can be solved by finding the Maximum Bipartite matching.

Complexity: $O(N^2)$

Problem H. Amina and her Patients

There are many solutions to this problem using the standard libraries. One of them is to create an appointment data structure and multiply the emergency level by -1 (and print the absolute value of it at the end) then use the standard sort function to sort the array of appointments.

Time Complexity: $O(Nlog(N))$

Problem I. Division Game

First note that each number can be decomposed into a product of prime numbers and its a unique decomposition. So this mean that the set of numbers you have to choose from is fixed. This will lead to Nim Game. You need to find the powers of each prime factor and then xor all the powers. The first player is in a winning position if the xor of all the powers is not 0.

Problem J. Powerful Strings

An optimal permutation must look like a unimodal function. It should be non-decreasing then non-increasing. More formally, there should not exist any three indices $i < j < k$ such that $S_i > S_j$ and $S_j < S_k$. (e.g. aaabbbcc...ccccba)

What remains to prove is the skewness of the distribution. The maximum power occurs when the string is symmetrical. In other words, if there are cnt_A occurrences of character A, then it is optimal to put $\lfloor \frac{cnt_A}{2} \rfloor$ on one end of the string, and $\lceil \frac{cnt_A}{2} \rceil$ on the other.