Problem A. Anime Creator

First, let's formalize the statement: There is a tournament on n vertices, with a cycle $1 \to 2 \to \ldots \to n \to 1$», from which we can find out the directions of the edges, as well as take some edge and delete the vertex it leads to. Our task is: removing vertices one by one and keeping the strong connectivity of the tournament graph, leave 3 vertices in it.

Let's note that the tournament graph remains strongly connected during all actions, so if we want to remove some vertex v, then we can always find an edge that leads to it. Suppose that at some time we know the Hamiltonian cycle $p_1 \to p_2 \to \ldots \to p_n \to p_1$, then if some edge is oriented towards $p_{i-1} \to p_{i+1}$, then we can remove the vertex p_i and construct a new Hamiltonian cycle on the remaining edges. Then, when we evaluate the number of requests, we will match each request to some removed node, so in the case above we will match the edge $p_{i-1} \to p_{i+1}$ to the removed node p_i.

Then let's check one by one the orientation of the edges $p_1 - p_3$, $p_3 - p_5$ and so on. If at some time the edge is oriented towards the cycle $p_i \to p_{i+2}$, then we simply delete the vertex p_{i+1} and continue the process until it ends at the last vertex or at the vertex right before last.

Case 1. If we ended at the last vertex, then our graph had an odd number of vertices, then consider the edge $p_1 - p_{n-1}$. If it is oriented as $p_2 \to p_{n-1}$, then a new Hamiltonian cycle $p_n \to p_{n-2} \to \ldots \to p_3 \to p_1 \to p_{n-1} \to p_n$, and the vertices $p_3, p_5, \ldots, p_{n-3}$ are such that they have one incoming edge from the cycle and one more to the cycle. Thus, these vertices can be removed one by one, and the graph will remain strongly connected, all edges, except for two, can be assigned to the removed vertices. Generally speaking, these edges can be assigned to some of the removed vertices so that each removed vertex is assigned to two edges, the only exception is the case $n = 5$, where one edge remains unmatched. In case the orientation is different – $p_{n-1} \to p_1$, we will remove the vertex p_n and assign it with two edges – $p_{n-1} \to p_1$ and $p_{n-2} \to p_n$ and thus proceed to the case 2.

Case 2. If we ended the path at the vertex right before last one, then n is even, and we will check the orientation of the edge $p_{n-1} - p_1$, if it turned out that $p_{n-1} \to p_1$, then like the previous case we will construct a cycle $p_{n-1} \to p_{n-3} \to \ldots \to p_1 \to p_{n-1}$ and remove the remaining half of the vertices by assigning them the edges of the cycle. Otherwise, remove the vertex p_n by assigning to it the edge $p_n \to p_{n-2}$ and go to the case 1.

By the actions described above, we will sooner or later come to the case $n = 4$, in which it is enough for us to check, for example, the edge $p_1 - p_3$ and in any case we will be able to remove either the vertex p_2 or p_4. If for $n = 5$ we still have an unmatched edge, then we compare it to the just removed vertex, together with the edge $p1 - p3$. Thus, we will remove $N - 3$ vertices, where each vertex will correspond to no more than 2 edges, which means that in total we will check no more than $2N - 6$ edges.

Problem C. Cotangent

From the Dedekind's theorem we can see that $l \cdot S(k, l) + k \cdot S(l, k) = ((k^2 + l^2 + 1)/3) - k \cdot l$. Because cotangent is periodic, then $l = l \bmod k$ is decreasing (like as in Euclidean algorithm.

So the time complexity is $O(log(k))$

Problem D. Distinct Numbers

Approximately in 2/3 queries segments are crossed. In this case problem is just classic "count of distinct numbers on segment and can be solved in $O((n + m) \log n)$ with rsq.

Otherwise, answer to the query is distinct in first + distinct in second - common in both. Distinct can be solved as above. Lets solve "count of common numbers on two non-intersecting segments".

Array has random values for a reason, lets look on statistic of frequencies of elements:

$\approx 0.36n$ elements have 0 occurrence in array

$\approx 0.36n$ elements have 1 occurrence in array

$\approx 0.18n$ elements have 2 occurrence in array

≈ 12000 elements have 3 occurrence in array (in max n)

≈ 3500 elements have ≥ 4 occurrence in array (in max n)

Obviously, cases 0 and 1 is not interesting.

Case 2 is can be transformed: for each element lets make 2D point $x = pos_1$ and $y = pos_2$, for each query (l_1, r_1, l_2, r_2) we interested in how many points contains rectangle $x = l_1 \dots r_1$ and $y = l_2 \dots r_2$. This problem is also well-known and can be solved in $O((n+m)\log n)$ with scanline+rsq.

Case ≥ 4 have not so much elements. Lets chunk all elements to parts of 64 elements. For each part make array $t_i = 2$ in some power $(0 \dots 63)$ such $t_i = t_j$ iff $a_i = a_j$. Lets build segment tree with operation "bitwise OR"in this array. Now if we want to answer the common-count-query, just ask OR on both segments and check bitcount of their bitwise AND. If we assign $T = count\_of\_numbers\_in\_case\_4$ than time complexity will be $O(T/64 \times (n + m\log n))$. With bottom-up segment tree and because of randomness of queries this part runs very fast.

In case 3 for each triple of positions (p_1, p_2, p_3) lets generate all three 2d points: (p_1, p_2), (p_1, p_3) and (p_2, p_3). Solve problem as in case 2. It can be happen that some of elements in current case counted twice. It can be only in two following situations:

1) $l_1 \leq p_1 \leq p_2 \leq r_1$ and $l_2 \leq p_3 \leq r_2$

2) $l_1 \leq p_1 \leq r_1$ and $l_2 \leq p_2 \leq p_3 \leq r_2$

For each query count of such triples needed to be subtracted from count-of-common.

Situations are similar by mirroring, so lets solve first one.

Lets iterate $i = 1 \dots n$ and at index i we want to answer on all queries with $r_1 = i$.

Lets maintain data structure that supports following operations:

1) add 2D point;

2) remove existing 2D point;

3) Get count of points in given rectangle.

When $i = p_2$ of some triple lets add point($x = p_1$, $y = p_3$).

When $i = r_1$ of some query lets ask of count of points in rectangle($x = l_1 \dots r_1$, $y = l_2 \dots r_2$).

When $i = p_3$ of some triple then remove point($x = p_1$, $y = p_3$) (after all above events in current i).

This problem is also well-known, but maybe not so popular, following is tutorial for it.

Lets build segment tree with vectors (also called merge-sort tree). Lets get all points that can be added during iterations, sort them by x and make segment tree that keep sorted y-s in each node. In node besides vector we need rsq data structure that keeps 1 next to y iff this point exists.

Now, adding/removing point query is changing element in each rsq for all corresponding nodes in tree. Asking for count of points in rectangle is classic segment tree query with rsq query in covered nodes. Each operation works in $O(\log^2 n)$

Problem F. Fruitland

First, let's calculate the minimum amount of coins to spend to move from one town to another. It can be done using Floyd Warshall algorithm ($O(N^3)$)

Then let's calculate the following dp:

$dp[i][j]$ is the minimum amount of money left when you are in city j while the cities from the bitmask i are visited.

$O(2^N \cdot N^2)$

For each of N cities, we can use the knapsack, i.e. $dp2[i][j]$ is the maximal value that can be reached using k coins and the fruits up to i-th.

That give $O(N \cdot K \cdot Y)$ time complexity.

Half full list

Divide the town into N / 2 groups A and B. In each group, first find the maximum satisfaction that can be obtained when using the j yen in the state of dp [i] [j]: = i (bit).

O (2N / 2 * Y * Y)

If you decide the bit of group A, decide the bit of group B, and decide the amount of money that the people of group A will spend on sweets, you will know the amount of money that people of group B can use for sweets.

O (2N / 2 * 2N / 2 * Y)