Problem A (Maiden Over)

Setter: Mohammad Ashraful Islam
Tester:
Alter: S.M. Shaheen Shah
Category: Adhoc

Problem B (All About Constraints)

Setter: Md Shafiul Islam
Tester: Mohammad Solaiman
Alter:
Category: Greedy, Construction

Editorial:

It is always optimal to use numbers which are powers of 2. There are only 64 such numbers. So if a range's length is greater than 64 then there is no answer.

Now we will greedily put numbers in ascending order as we have to find the lexicographically smallest array. For that, we will sort the constraints by smaller L.

Assuming there is three ranges [2 7], [3 6], [3 10] and N = 10

At first for position 2 to 7 we will put [1, 2, 4, 8, 16, 32]

So our array looks like this

_, 1 , 2 , 4 , 8 , 16 , 32 , _ , _ , _ [_ represents no number yet]

Next comes the [3 6], As all the position is already covered by [3 6], we can safely ignore this constraint as it is already satisfied.

For the last range position 4-7 is already has a number and we will keep it that way and put 1 64 128 in position 8-7. We cannot use 2, 4, 8, 16, 32 again. And put 1 in the blank positions. So final answer is

1 , 1, 2, 4, 8, 16, 32, 1, 64, 128

Problem C (Some Game)

Setter: Tanzir Pial
Tester: Rafid Bin Mostofa
Alter: Ashiqul Islam
Category: Alpha-Beta Pruning, Combinatorics

Let F(n,c) be the number of ways n distinct balls that can be linearly arranged in k indistinguishable boxes.

F(n,c) = nCr(n-1, c-1) * n! / c!

This can be found out using techniques similar to stars and bars analogy. We can place the n balls in n! Different ways and there will be n-1 gaps between them. From these n-1 gaps, we pick c-1 gaps in nCr(n-1,c-1) ways for placing bars to denote c non-empty boxes. Finally we need to divide by c! Since we are overcounting the arrangement of the boxes.

Now that we know a O(1) formula for solving F(n,c), all that remains is the backtrack solution for the minimax game. A naive brute force will be O (2^P). Due to the random nature of the input, a backtrack with alpha beta pruning will be average case O(2^(.75P)). *

F(n,c) has a O(n*c) dp solution too but that will require too much memory and time.

Bonus: Is it possible to create adversarial input for a solution using alpha beta pruning for this problem?

* In general, for a backtrack with b branching factor and d depth, alpha beta pruning brings the complexity down to O(b^(.75d)) average case runtime for random values at leaf nodes.

Problem D (Maxxxxxximum Spanning Tree)

Setter: Rezwan Mahmud
Tester: Rafid Bin Mostofa
Alter:
Category: Graph, Number Theory
Only keep those edges(i, j) where j is a multiple of i. You will have O(n ln n) edges in total for consideration. All the other edges can be discarded(Proof is left as an exercise to the reader). Now you can run Kruskal's algorithm. Note that you have to do counting sort to sort the edges in order to fit your solution under the TL.

Problem E (Number of Zeros)

Setter: Anindya Das
Tester: Hasnain Heickal
Alter: Aminul Haq
Category: Number theory, Combinatorics

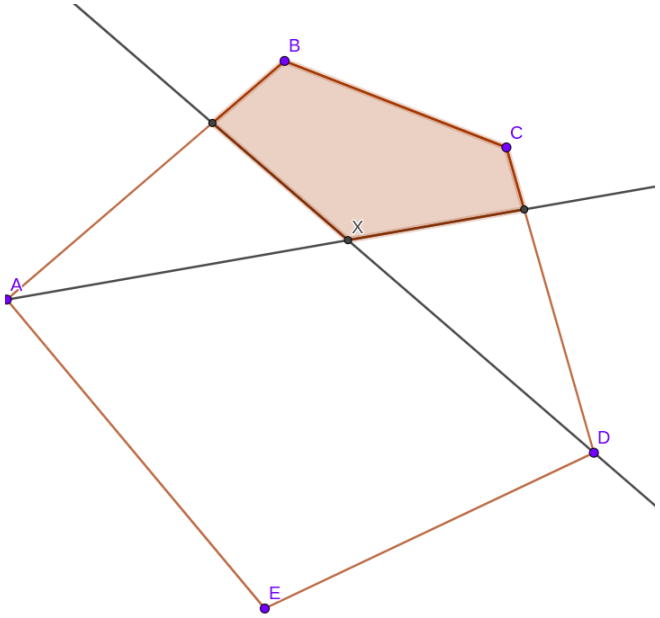
First we need to find the formula for V in terms of ai. We can observe that it would be the product of ai ^ (n-1)C(i-1). Now you just need to find out the number of zeros in factorial(i) and multiply it by (n-1)C(i-1) (use the formula: nCr = n! / (r! * (n-r)!), precalc all factorials and their modular inverses upto 10^5) and sum it over all i.

Problem F (Unbalanced Polygon)

Setter: Pritom Kundu
Tester: Muhammad Ridowan
Alter: Nafis Sadique

Tags: Geometry

First, we will find a characterization for T(i). Let X be the center of all vertices except P_i. Then T(i) is the area of the region of the polygon bounded by the two lines $P_{i-1}X$ and $P_{i+1}X$. See the figure for clarity. The proof is left as an exercise to the reader.



Region of Unbalanced positions for point E.

After this characterization is found, the task left is to calculate this area for each vertex efficiently.

First we find which side of the polygon $P_{i-1}X$ intersects with. Note that the line divides the polygon into two regions. The intersecting side is the one, for which the two endpoints are on opposite sides of the polygon. Thus we can find the furthest point on the left side with a binary search. Then we can easily find the intersection of $P_{i-1}X$ and this side. Let's call this point Y.

Similarly, we find which side of the polygon $P_{i+1}X$ intersects with. Then we can easily find the intersection of $P_{i+1}X$ and this side. Let's call this point Z. This part takes O(log n) time per vertex.

Then we can have determined all vertices of the region and can calculate the area. Note that each region has at most four sides that are not also sides of the given polygon. On the other hand, it can be proven that each polygon side can be strictly inside at most two regions. The two facts together prove that there are no more than 6n sides over all n regions. Thus the area calculation can be done in brute force in O(n) amortised time.

Alternatively, a prefix-sum like technique can be used to calculate the area in O(1) time per vertex. Overall time: O(n log n)

Problem G (Polymorphism)

Setter: Aminul Haq
Tester: S. M. Saheen Sha
Alter: Imran Bin Azad
Category: DFS, Sparse table
Solution: First, let's understand the problem, it says you have a root node of a tree which is the "Main" class and then in each query you either add a new node to the tree or answer whether a given node is an ancestor of another given node or not.
There are two soutions, one is to find LCA using sparse table and the other one is to solve the problem offline with a simple DFS. Let's discuss the second one. For this, first, we will read all the queries, and build the tree, and remember "Main" class is the root of the tree. Then we can start a DFS from from the root and keep track of all the ancestor nodes, for that we can use a Set. And then when we visit a node, then we can check all the queries on it.
Complexity: O(N + Q) where N is the number of nodes, and Q is the number of queries.

Problem H (Shift Digit)

Setter: Sabit Zahin
Tester: Rafid Bin Mostafa
Alter: S.M. Shaheen Sha
Tags: Ad hoc, DS, Simulation
Solution:
There are two key points to solve this problem.

1. Number of Digits is Only 10
2. If a digit shifts, that will form a group of the same digits and will stay together for the remaining operations.

Now, coming to the solution. To solve this Problem, we will maintain two lists. First list will contain digits in the given order. When we shift a digit we will move this digit to the end of the Second List, and delete that digit from the first list, the rest of digits will remain the same with the given order. Now think, In the second list, do we need to store all the digits ? No. We will store only that digit and frequency. If the digit is already in the second list, we will change the order only in that list. After every operation we have to truncate the leading zeros if any.
Now calculating the value, now observe that the first list will change at most 10 times. So we can calculate the value for this list by iterating all indexes after each change in that list. Suppose there are M digits (d1,d2,.. dm) in the first list and S = d1 * 10^(M-1) + d2 * 10^(M-2) + … +dm^ 10^0

For the second list, suppose there are N digits (x1, x2, …. xn) and these frequencies are (f1,f2,..fn)
We can pre-calculated this,
S1 = x1*10^0 + x1*10^1+....x1*10^(f1-1)
S2 = x2*10^0 + x2*10^1+....x2*10^(f2-1)
…
Sn = xn*10^0 + xn*10^1+....xn*10^(fn-1)
Then Final value will be, = S*10^(f1+f2+...+fn) + S1*10^(f2+f3+..+fn) + … + Sn

Problem I (Seven Segment Display)

Setter: Hasnain Heickal
Tester: Md Mahamudur Rahaman Sajib
Alter: Muhiminul Islam Osim, Abdullah Al Maruf
Tags: Dynamic Programming

Suppose two numbers S and T are given. We need to make T from S using minimum number of moves. Let's try to solve this problem first.
a = number of sticks which are at the same position in both T and S.
b = number of sticks which are available in T but not in S.
c = number of sticks which are available in S but not in T.
It is very easy to see that b = c and we can say that we are moving only c number of sticks. So the minimum number of moves is b = c.
Now let's try to solve another problem. Given S, we need to make a number T such that the minimum number of moves are minimised. We just need to find that minimised minimum number of moves. Let's try to solve the problem using dynamic programming and we will use this dp information to solve the actual problem.
dp[i][j] = minimum number of sticks which are in T but not in S to cover [i : n - 1] position using j number of sticks(as we said above minimum number of moves is b = c, so we are trying to minimise b).
cost(S[i], d) = number of sticks which are available in digit d but not in digit S[i].
dp[i][j] = min(cost(S[i], d) + dp[i + 1][j - stick_count(d)]) [0 <= d <= 9]
Now we can use this dp table to solve the actual problem as dp[i][j] is equivalent to the minimum number of moves.
In this problem we need to build a maximum number using k number of moves. So let's try to solve this problem greedily. We will try to put the maximum digit in the most significant position. For example we already filled up [0 : i - 1] positions correctly and we used x number of sticks and y(it can be easily found with the above theory) number of moves already. Now we want to put a digit in the i'th position so that the number is maximised. We will try all the digits, d from 9 to 0 and check the dp[i + 1][total_stick - x - stick_count(d)]. If y + cost(S[i], d) + dp[i + 1][total_stick - x - stick_count(d)] <= k, then we can say that digit d is a valid digit. We will put the maximum valid digit. So time complexity of building the dp is O(n * n * 10) per test case and time complexity of query solving part is O(q * n * 10) per test case. Total time complexity O(t * (n * n * 10 + q * n * 10)).

Problem J (K Subsequence Problem)

Setter: Md. Nafis Sadique
Tester: Sabit Zahin
Alter: Pritom Kundu
Tags: Combinatorics, FFT, Divide and Conquer

Let's think about each unique integer that occurs in the array. If it occurs c times, then we can include at least one occurrence of it in our subsequence $2^c - 1$ ways.

Let $a_1, a_2, a_3, \cdots, a_p$ be the unique elements in the array. Let c_i be the number of time a_i occurs in the array. We define the polynomials $P_i := 1 + (2^{c_i} - 1)x$. Consider the product of these polynomials $P = P_1 P_2 \cdots P_p$. The coefficient of x^k in this polynomial is our required answer.

Note that, P has degree at most n. Thus the coefficients of P can be calculated by divide and conquer and polynomial multiplication (with fft). Overall complexity: $O(n \log^2 n)$

Problem K (Change the username)

Setter: Raihat Zaman Neloy
Tester: Nafis Sadique
Alter: S. M. Shaheen Sha
Tags: AdHoc, STL (map/set)

This problem is solvable using two maps or one map and a set. Whenever the first operation appears, we just need to check if a user is already using the username Y. Then the change operation won't happen and at the same time we will store the information of X into our username database. If the change operation is successful, then we will delete info of X from our username database and keep X in another map or set which let us know about the usernames that are released because of operation 1. One thing we should keep in mind, when username X is released it will behave like a totally new username in future if ever used.

For operation two, we need to loop up for the username X if we have ever seen that before. If not, then we will print the X as output and at the same time store it's information in the database. Otherwise if we find X in released database information, we will print "Not in Use!", otherwise the expected answer we were keeping track of.