Analysis

NSTU IUPC 2020

A. City of Burgerland

Setter: Emrul Chowdhury
Tester and alternate solution writer: Anik Sarker, Tahsin Masrur
Techniques: Data Structure

Find the number of shops numbered from l to r with capacity less than cap and sum of their capacities. Let number of such shops as num and sum of their capacities as sum, then we need to increase capacity by $cap \times num - sum$. You can use various data structures to find those two values. Among them, most of the solution were using Merge Sort Tree.

B. Joker's GCD Test

Setter: Anik Sarker
Tester: Emrul Chowdhury
Techniques: Mobius Inversion, Combinatorics, Number Theory

Lets solve the counting version of the problem without any updates. We need to calculate the sum of gcd over each pair of elements in P. Let M be the maximum element in P. Then using **Mobius Inversion**, this sum can be formulated as $\sum_{v=1}^{M} \phi(v) * Count(v) * Count(v)$.
Here, $Count(v)$ = number of elements in P which are multiples of v.
We can learn more details on **Mobius Inversion** from **here**.

We initially pre-compute divisors for all numbers in [1, M] in $O(M \log M)$ via a sieve-like algorithm. Then for each number P_i in the array and for each divisor v of P_i, we increase $Count(v)$ by exactly 1. This gives us $Count(v)$ value for each v in $[1, M]$, in total $O(\sum_{i=1}^{n} d(P_i))$ time.

For handling updates, when we add an element X_i to the array, we need to increase $Count(v)$ by exactly 1 for each divisor v of X_i and modify their contribution. Similarly, when we delete an element X_i from the array, we need to decrease $Count(v)$ by exactly 1 for each divisor v of X_i and modify their contribution. This can be done in total $O(\sum_{i=1}^{q} d(X_i))$ time.

So overall complexity : $O(M \log M + \sum_{i=1}^{n} d(P_i) + \sum_{i=1}^{q} d(X_i))$.

C. Restore Missing Values

Setter: Md Abdul Mazed
Tester and alternate solution writer: Anik Sarkar
Techniques: Segment tree

First, you need to know the start index and end index of the array of all values from 1 to Q. For missing values you will not get any start or end index. If there is no missing value and last value Q is not found in the array then you can't make all query be succeeded. As you need to choose the lowest value for the missing values, update the range [1,N] with 1 at first query. Now for i-th query update the range from start index and end index of i in the array. For these update, you can use update functionality of segment tree. After Q queries check the array again. If you find any value that wasn't missing in given input, is not same as given input, then you can decide that the array can't be restored. Otherwise you can get expected array.

Time Complexity: O($T * n * log(n)$)

D. Beauty Factor

Setter: Shaon Kanti Dhar
Tester and alternate solution writer: Tahsin Masrur
Techniques: Prime factor, Cumulative sum, Binary search

Here maximum number of $X + N - 1 <= 10^5$ and maximum number of distinct prime factor from 1 to 10^5 is 6. So, pre-calculate the count of distinct prime factor from 1 to 10^5 and store them in a 2D array based on their number of distinct prime factor which is in sorted order and also calculate their cumulative sum.
Now for T queries use Binary Search Technique and find the result for given L to R.

Time Complexity: O($T * 6 * log(n)$)

E. Find the Good Sequence

Setter: Shaon Kanti Dhar
Tester and alternate solution writer: Pranto Chandra Saha
Category: Greedy, Implementation

Answer for the last element is always 0. So for each element from $N-1$ to 1 if it is a good number then $answer =$ `previous_answer` $+ 1$. If the previous answer is 0, then $answer = answer + 1$.

Time Complexity: O(n)

F. Lexicographical Smallest String

Setter: Asaduzzaman Nur Shuvo
Tester and alternate solution writer: Emrul Chowdhury

$O(nk)$ solution

Just run a brute force. After finding which character should be deleted, we can generate k strings $A_1, A_2, ..., A_k$ where A_i refers to the string keeping ith occurence of the original string. Then output smallest string, $\min(A_1, A_2, ..., A_k)$.

$O(n)$ solution

After finding which character should be deleted, we keep that one occurrence of the character if the next character of the string is not the same and lexicographical Bigger. In other cases, we keep any of the occurrences of that character.

Note: In case of $k = 0$ or $k = 1$ no deletion takes place.

G. Jontrona of Liakot

Setter: Tahsin Masrur
Tester and alternate solution writer: Rafid Bin Mostofa
Techniques: Euler tour or DFS time, persistent trie

First find the starting time and ending time of all the nodes of tree using a DFS. Consider the array of time, the subarray starting at starting time of node U (denoted by S_U from now on) and ending time of U (E_U) represents the subtree rooteed at U. So basically, you need to find the K-th number among all the numbers in the subarrays $[S_U, S_V - 1]$ and $[E_V + 1, E_U]$ after XOR-ing them with Z. If we had to do this for a single array always, it'd be easy with Trie, a well-known problem. For our problem, we need to maintain persistent trie i.e. keeping a version of trie for each time in the array of DFS time. We can then just consider the four versions relevant to each query and walk down the tries accordingly to find the solution.

Time complexity: O($Qlog_2(10^9)$)
Space complexity: O($Nlog_2(10^9) + Qlog_2(10^9)$)

H. Robin's Birthday

Setter: Asaduzzaman Nur Shuvo
Tester: Anik Sarker
Techniques: Giveaway

Just check whether two adjacent elements of the array/sequence are same.

I. Sgt. Laugh

Setter: Rafid Bin Mostofa
Tester: Tahsin Masrur
Techniques: DP

Let $dp(n, k)$ be the result for indices $[1, n]$ and sequence length k. Then,

$$dp(i, 1) = 0 \tag{1}$$
$$dp(n, k) = \min_{i<n,\ a_i<a_n} \{dp(i, k-1) + a_n - a_i\} \tag{2}$$

The obvious implementation will not run in time. We can create a BIT/Segment Tree using a_i as keys and $dp(i, k-1) - a_i$ as value. Then a query in range $[1, a_n - 1]$ should get the minimum for the transition mentioned above.

Complexity: $O(nk \log n)$

J. Earn Your Cupcake

Setter: Rayhan Chowdhury
Tester: Rafid Bin Mostofa
Techniques: Probability, Counting, DP

For an $N \times N$ grid, we can choose 4 points in $\binom{(N+1)^2}{4}$ ways. Also, we can count the number of unique rectangles in this grid using DP; details of which can be found **here**.

Let's define a rectangle invalid if any of its corner points don't contain a cupcake. We can use Principle of Inclusion Exclusion to count the number of such invalid rectangles.

$$\begin{aligned}\text{No. of Invalid rectangles} = &\text{ No. of rectangles with at least 1 invalid corner } (P_1)\\ &- \text{No. of rectangles with at least 2 invalid corners } (P_2)\\ &+ \text{No. of rectangles with at least 3 invalid corners } (P_3)\\ &- \text{No. of rectangles with 4 invalid corners } (P_4)\end{aligned}$$

For P_1, we can iterate over the given invalid points and for each of those points we can iterate over every point in grid that produces a positive length arm for a rectangle. For this arm, we calculate the number of rectangles possible in this grid. For an arm with points $(x_1, y_1),\ (x_2, y_2)$ this can be calculated via the slope of its perpendicular line.

For P_2, we can take two invalid points and consider them as an arm for a rectangle and count the number of such rectangles. However, the invalid points can be placed on a diagonal and that must be handled. For two points $(x_1, y_1),\ (x_2, y_2)$ on diagonal, we can count the number of such rectangles

by calculating the other two possible corner points. For any such point (x, y), the following must hold:

$$\frac{y_1 - y}{x_1 - x} = -\frac{x_2 - x}{y_2 - y} \tag{3}$$

Thus,

$$x^2 + y^2 - (x_1 + x_2)x - (y_1 + y_2)y + x_1x_2 + y_1y_2 = 0 \tag{4}$$

We can count such pair of corner points via iterating over every x coordinate in the grid.

For P_3 and P_4, taking 3 and 4 invalid points and checking whether they construct a rectangle respectively is enough.

Finally our result would be

$$\frac{\text{No. of Rectangles } - \text{No. of Invalid Rectangles}}{\binom{(N+1)^2}{4}} \pmod{10^9 + 7} \tag{5}$$

Time complexity for finding number of invalid rectangles: $O(M^4 + M^2N)$.

Implementations can be found **here**.