Fuzzy Uniqueness

Rahim Mammadli

December 2021

1 Subtask 1

For this subtask it is enough to go through each $(i, j), 1 \leq i \leq j \leq n$ and count the number of unique elements for contiguous subarray starting at i and ending at j. One can use map/unordered_map in $C++$ to solve this subtask. Expected time complexity : $O(n^3 * log(n))$ or $O(n^3)$

2 Subtask 2

Let us fix the starting point i, and find the uniqueness for each ending point j. As we sweep for j from left to right, if we come across a number we haven't seen before we add 1 to our uniqueness count. If we come across a number we have seen exactly once before, then we subtract 1 from our uniqueness count. While doing all this, we keep track of contiguous subarrays whose uniqueness is between l and r. Expected time complexity : $O(n^2 * log(n))$ or $O(n^2)$

3 Subtask 3

There is only $d = 500$ distinct elements, and we only need to count subarrays whose uniqueness is exactly 1. Let us fix the starting point i. Then for at most 500 numbers we will have ranges (x_i, y_i), where x_i denotes

the index of first occurrence of number i that is greater than or equal to i, and y_i denotes the index of second occurrence of number i that is greater than or equal to i. Put $+1$ at each x_i, and -1 at each y_i. For given i, we are only interested in number of points j whose prefix sum , based on the given $+1$ and -1s, is exactly 1. This can be counted by a simple sweep in $O(d)$. The $+1$ and -1s can be kept in a *map* and updated at each index accordingly. This map will cost $O(n * log(n))$ in total. Expected time complexity : $O(n * d)$.

4 Subtask 4

Once again, we will sweep from left to right for starting indices i, and then count number of ending indices j for which the uniqueness count is 0 (as per given subtask). Let us denote the number at index m as a_m. Let us denote pv as the last occurrence of a_m to the left of m, that is $pv = \{max(l)$, such that $l < m$, and $a_l == a_m\}$. Similarly let us denote nx as the first occurrence of a_m to the right of m, that is $nx = \{min(l)$, such that $l > m$, and $a_l == a_m\}$. This number will result in adding $+1$ to some contiguous subarrays of the given array. This subarrays can be generalized as : subarrays whose starting point i is in range $[pv + 1, m]$, and ending point j is in range $[m, nx - 1]$. Let these ranges represent our updates. As sweep from left to right for starting indices i, when we reach $pv + 1$ we will add $+1$ to each ending index j, such that $m \leq j \leq nx - 1$, and when we reach $m + 1$ we will add -1 to each ending index j, such that $m \leq j \leq nx - 1$. While doing all of these we need to keep track of number of ending indices whose uniqueness count is 0. This can be done with range addition, range min lazy segment tree. Have an extra variable to keep track of number of indices equal to range min. Note that Subtask 3, can be solved in the same way, with keeping a few extra variables inside each node of the segment tree. Expected time complexity : $O(n * log(n))$.

5 Subtask 5

We will sweep from left to right for starting indices i, and then count number of ending indices j for which the uniqueness count is between l and r. We will use the aforementioned ranges and updates. So we have range addition by 1, subtraction by 1, and we want to keep track of ending indices

whose uniqueness count is in between l and r. Divide the candidate ending indices into $sqrt(n)$ blocks each of size $sqrt(n)$, at each block keep n values. Let us denote this array of size $sqrt(n) * n$ as occ. $occ[i][j]$ stand for the number of ending indices in block i whose uniqueness value is currently j, we also have another array $lazy$, where $lazy[i]$ denotes the lazy sum added to the whole block i. With these structures the whole question can be solved in $O(n * sqrt(n))$ time, and $O(n * sqrt(n))$ memory. For a given range update, if only a part of a block is within the given update, go through the indices one by one, and apply the query. While doing that, update the count of candidate ending indices whose uniqueness value is between l and r, accordingly. For whole blocks that are inside the given range, go through them, add to their $lazy$ count, and use occ and $lazy$ arrays to update the count of candidate ending indices whose uniqueness value is between l and r, accordingly