D. Strange Queries

Editorial

The problem is stated very simple. You are given an array $A[1, \ldots, N]$ of N integers. You have to answer Q queries about the array. The answer for the i^{th} query is the number of pairs of indices (i, j), such that $L_1 \leq i \leq R_1$, $L_2 \leq j \leq R_2$ and $A_i = A_j$.

First, as often in problems with range queries, we will simplify a result of a single $\texttt{query}(L_1, R_1, L_2, R_2)$ to the result of $\texttt{query}(1, R_1, 1, R_2)$ - $\texttt{query}(1, R_1, 1, L_2 - 1)$ - $\texttt{query}(1, R_2, L_1 - 1)$ + $\texttt{query}(1, L_1 - 1, 1, L_2 - 1)$. We can do this, because a result for a single query is just the number of indices.

Since, $N \leq 10^5$, obviously we cannot use the a $O(N^2)$ method. However, N is small enough to use the method called **square root decomposition** of queries.

The first step in our solution, is to precompute the answer for some queries. Let's call K a block size. Then let $p_{i,j}$ be the answer for a query with the right endpoint of the first range at index $i \cdot K$, where $1 \leq i \cdot K \leq N$, and the right endpoint of the second range at any index $1 \leq j \leq N$. Notice that we can easily precompute the whole p table in $O(N \cdot K)$ time. This is true, because there are total of $N \cdot K$ entries in p, and the value of $p_{i,j}$ can be computed from $p_{i,j-1}$ in constant time by using counters of values implemented as an array. This is possible since all integers in A are within a range $[1, N]$.

So far so good, we can answer queries where one right endpoint is arbitrary and the second one is a multiple of K. How can we use these precomputed values to answer each query fast enough? We can decouple a single query into a few queries for which we have precomputed answers, and one query which is so small that we can answer it quickly.

In more details, let's consider a single $\texttt{query}(1, c_1 \cdot K + r_1, 1, c_2 \cdot K + r_2)$. We can notice, that the result for this query can be computed as $\texttt{query}(1, c_1 \cdot K + r_1, 1, c_2 \cdot K)$ + $\texttt{query}(1, c_1 \cdot K, 1, c_2 \cdot K + r_2)$ - $\texttt{query}(1, c_1 \cdot K, 1, c_2 \cdot K)$ + $\texttt{query}(c_1 \cdot K, c_1 \cdot K + r_1, c_2 \cdot K, c_2, \cdot K + r_2)$. Each of the first 3 of these queries has at least one right endpoint equal to a multiple of K, so its value is precomputed in p table. Both two ranges in the fourth query are not greater than K, so we can compute the answer it in $O(K)$ time by simply counting elements occurring in both ranges.

Based on the above observations, we can notice that answering a single query takes O(K) time, so the total complexity of this solution is $O(N \cdot K + Q \cdot K)$, and we can minimize it if we let $K = \sqrt{N}$. That is why this kind of approach is called a square root decomposition.