

Participants entry

Problem author: Alexey Mikhnenko

Problem developer: Artyom Agafonov

We will simulate the process. Let us iterate over the participants in increasing order of their numbers. If the color of the current participant is the same as the color of the previously released participant, then we put this participant into the queue. Otherwise, we mark that this participant is the next one to be released, and then release the participant waiting in the queue, if the queue is not empty. Finally, after we have processed all participants, we release all participants that are waiting in the queue. Thus, we obtain a linear simulation of the process, and therefore this solution works in $O(qn)$.

Notice that if, during such a simulation, the queue is empty and the color of the participant's T-shirt differs from the color of the previously released participant, then this participant will be released under his own number. Notice that all participants with numbers smaller than the current one do not affect the arrangement of this participant and all subsequent ones, so we may assume that for such participants the process starts anew, with their number taken into account.

Suppose we start the process from participant number i . How do we find the next participant from whom we can again assume that the process starts anew? Put $+1$ at the positions of participants whose T-shirt color is a_i and -1 at all other positions. Then the smallest index j such that the sum from i to j inclusive is 0 will be the desired one. For each position i , draw a directed edge to $p_i = j$. These edges form a directed forest.

Let us understand how to answer a query using this forest. Starting from the first participant, we will gradually follow the edges while the next participant has number at most y . Suppose we stop at i ; then, by the property above, we may assume that the process starts from participant i . Notice that when arranging participants on the segment from i to $p_i - 1$, positions $i, i + 2, i + 4, \dots, p_i - 1$ will be occupied by participants whose T-shirt color is a_i , and positions $i + 1, i + 3, \dots, p_i - 2$ will be occupied by all the others. Therefore, to determine the position of participant y , we need to find out how many participants with color a_i stand before him. For this, we can use binary search over the array of positions of participants with T-shirt color a_i . Such an array can be easily maintained under update queries.

Thus, the problem is split into two parts: we need to dynamically maintain this forest when the colors of two adjacent participants change, and quickly find the edge covering position y . The constraints in some subgroups allowed simplifying the implementation of these parts of the solution. Below, we present the solution for the full score, using link-cut trees. Another option is root decomposition; with careful implementation, it also allowed earning full score in this problem.

The second part, which consists of finding the covering edge, is implemented by the expose operation from vertex 1, which makes it possible to extract the path containing all participants from whom the process will start anew. The participant of interest can then be found by descending the resulting splay tree.

If we determine for which participants and how the values p_i change, then these changes can be maintained in the tree using link and cut operations. Notice that for colors different from a_x and a_{x+1} , both participants contributed -1 to the corresponding sums. Therefore, swapping them does not affect the sums. If the colors a_x and a_{x+1} are the same, then nothing changes. Consider the case $a_x \neq a_{x+1}$. First, the outgoing edges from x and $x + 1$ will change. Notice that for color a_x , the contributions of positions x and $x + 1$ changed from $+1, -1$ to $-1, +1$. Now, when computing the sum starting from some position i of color a_x , the algorithm may obtain 0 at position x . This means that previously the sum up to position x was equal to 2. Knowing this, we can find the position where the prefix sum for color a_x turned out to be 2 less than at x . For example, this can be implemented by descending a segment tree for color a_x , where the leaf i stores the sum on the prefix up to the i -th participant with T-shirt color a_x (note that it is enough to store such sums, since between such participants the prefix sums decrease linearly). This position will be the one to which the edge would have pointed, and now we must redirect it to x . By the way the edges are drawn, there will be exactly one such edge. Similarly, we can analyze the case for color a_{x+1} — from it we will also obtain at most one candidate whose value in the array p changes. To find the

position to which the new edge will point, we need to make one more query in the corresponding segment tree to find the first occurrence after position i of a prefix sum that is 1 smaller than the prefix sum before position i .

Thus, we obtain a solution that works in $O(q \log n)$. Also, for full score it was possible to implement solutions working in $O(q \log^2 n)$, where the splay tree in the link-cut tree is replaced by a treap, or in $O(q\sqrt{n})$, where the whole array is divided into blocks of size \sqrt{n} and for each element we maintain a compressed jump that points to the first element outside the current block that can be reached by moving upward along the edges. Then, for an update query, we need to update this jump for elements from at most 4 blocks, and the search for the covering edge is performed by following such compressed jumps until the block containing the query is reached.