Problem E. Longest Unfriendly Subsequence

Author: Anton Trygub
Developer: Anton Trygub
Editorialist: Anton Trygub

Subtask 1. Any subsequence of such a sequence is nondecreasing. Unfriendly sequences, however, do not allow equality of two adjacent elements, so any unfriendly subsequence of this sequence has to be **strictly** increasing. This means, that each value will appear in such a subsequence at most once.

But then we can delete duplicates and take a subsequence containing precisely one occurrence of each element that appears in a. As all elements of this subsequence are distinct, it's unfriendly. So, the answer for this subtask is just the number of distinct elements in a. We can find it in $O(n)$.

Subtask 2. We can just consider all possible $2^n - 1$ nonempty subsequences of a, check each for unfriendliness in $O(n)$ time, and output the length of the longest unfriendly. This takes $O(2^n n)$ time. As $t \leq 10^5$, this easily fits in TL for $n \leq 8$.

Subtask 3. Clearly, for $n = 1$ answer is 1, and for $n \geq 2$ it's ≥ 2 (as any subsequence of length exactly 2 is unfriendly).

Let's use dynamic programming. Let $dp[i][j]$ for $1 \leq i < j \leq n$ denote the length of the longest unfriendly subsequence of a, in which the last element is a_j, and the second last is a_i. If $a_i = a_j$, $dp[i][j] = 0$. Otherwise, $dp[i][j] = max(2, max_{1 \leq k < i} dp[k][i] + 1))$ over k for which $a_k \neq a_i$ and $a_k \neq a_j$. We can calculate this dp table in $O(n^3)$ for a single test case, which is fast enough.

Subtask 4. Let's look at any unfriendly sequence $b_1, b_2, \ldots, b_m$ such that for all i $1 \leq b_i \leq 3$. Each 3 consecutive elements of b are distinct, therefore b_i, b_{i+1}, b_{i+2} are some permutation of $1, 2, 3$ for $1 \leq i \leq n-2$. Then, however, $b_{i+1}, b_{i+2}, b_{i+3}$ also are such a permutation. As b_i and b_{i+3} both differ from two distinct values (b_{i+1}, b_{i+2}), they must be equal. So, $b_i = b_{i+3}$ for each i; b has to be periodic with period 3.

Then, just try each possible start of the subsequence b p_1, p_2, p_3 — every permutation of $(1, 2, 3)$. For each of them, take elements $p_1, p_2, p_3, p_1, p_2, \ldots$ as soon as you see them. Output the largest answer over these 6 options.

Subtask 5. Let's go through our sequence a from left to right and keep the following dynamic programming table: let $dp[x][y]$ denote the length of the longest unfriendly subsequence of a up to this moment, whose last element is y, and second last element is x. Initially, we can set each value in this table to $-INF$ (where $INF = 10^9$, for example). Let's also keep track of what elements have already appeared in our sequence.

It turns out that it's easy to update this table: when we are at position i, we just need to update the values of $dp[x][a_i]$ for each $x \neq a_i$. If x hasn't appeared before, there is no subsequence ending with (x, a_i), otherwise, do $dp[x][a_i] = max(dp[x][a_i], 2)$. Then, we need to do $dp[x][a_i] = max(dp[x][a_i], dp[y][x]+1)$ over all $y \neq x, a_i$. Updating this table after seeing the next element takes $O(MAX^2)$, with overall complexity $O(MAX^2 n)$ per test case, which fits easily.

Subtask 6. Let's modify our algorithm from **Subtask 5** a little. Clearly, we can assume that elements are in the range $[1, n]$ (just map k-th smallest value to k, we don't care about the exact values of elements, we only care about which elements are equal to which). Now, again, let's keep $dp[x][y]$ for $x \neq y$: the length of the longest unfriendly subsequence of a up to this moment which ends with (x, y). The difficulty lies in updating $dp[x][a_i] = max(dp[x][a_i], dp[y][x] + 1)$ over all $y \neq x, a_i$: this can take $O(n^3)$, which for $n = 10000$ has no chance of passing.

But let's note that we don't actually need **all** the values $dp[y][x]$ to update this table. We need the largest value among the ones for which $y \neq a_i$. Then, for each y let's keep two values $x_1 \neq y, x_2 \neq y$, such that the values $dp[x_1][y], dp[x_2][y]$ are the largest among all $dp[x][y]$. Then, we would just have 2 (at most) candidates to check. After we do this for each y, we will recalculate the best choices for the previous element for a_i.

This way, processing new element takes $O(n)$, and the entire algorithm runs in $O(n^2)$ time, which passes easily.

Subtask 7. For this subtask, we will have to analyze the structure of the longest unfriendly subsequence a bit more.

Consider the longest unfriendly subsequence of a. Suppose that it contains a_i. What could be the previous element before a_i, if there is any? Clearly, if it's some value x, it's optimal to take the last occurrence of x before a_i.

What we did in previous subtasks was going through all possible candidates for x. However, as it turns out, we don't need that many. Among all last occurrences of elements before a_i, consider 5 rightmost (if there are at least 5). Suppose that we don't take any of those as our x. Then, I claim, we can extend our unfriendly subsequence by inserting one of these rightmost 5 last occurrences into it.

Indeed, two (or less, if there are less than two) elements to the left of a_i in this subsequence, a_i, and the element to the right, if there is any. They are the only prohibited values for the x (if we want to insert x right before a_i in this subsequence). Then one of those 5 last occurrences would not be prohibited, and the subsequence wouldn't be the longest possible.

So, for each a_i, we know the set of at most 5 possible candidates for the previous element in the longest unfriendly subsequence. Therefore, we can once again use dynamic programming of form $[cand][last]$, indicating the length of the longest possible unfriendly subsequence, ending in (a_{cand}, a_{last}). For each $last$, we have at most 5 cand. So, when processing new $last$, we need to do just $MAGIC^2$ checks (where $MAGIC = 5$).

We can keep this dp in maps, and keep the last occurrence of each element with a simple set. The total complexity is $O(n(5^2 + \log n))$.