More Gifts

Subgroup 1. $n \leq 100, k \leq 10$.

We use dynamic programming. Let $dp[i]$ be the minimum number of subsegments in a prefix of length i, with $dp[1] = 1$.

Recalculation for state i: $dp[i] = \min(dp[i], dp[j] + 1)$, if the subsegment $[j + 1, i]$ contains no more than t different numbers.

If we recalculate this dynamic from the end, we can check the condition in $O(1)$, maintaining the number of different numbers in the subsegment. Thus, we get a solution for $O((nk)^2)$ or $O((nk)^2 \log(nk))$.

Subgroup 2. $t = 1$.

We divide the original array into blocks of consecutive equal numbers. Let the number of blocks be cnt. Then, the answer will be $cnt \cdot k$ if the first block does not match the last one, otherwise $(cnt - 1) \cdot k + 1$.

Proof of the Greedy Solution.

Consider some optimal partition, let it differ from the greedy one. Find the first position where there are differences. In this case, the length of the subsegment in the optimal partition is less than in the greedy one. Shift the right boundary of this subsegment to the position in our greedy solution. The partition will remain correct and the number of subsegments will not increase. By continuing this process, the optimal partition will turn into the greedy one, and the number of subsegments will not increase. It follows that the greedy partition is optimal.

Subgroup 3. $n \leq 1000, k \leq 1000$.

We construct a fully repeated array of size nk. Using a greedy algorithm, we obtain the minimum number of gifts. Solution for $O(nk)$ or $O(nk \log(nk))$.

Subgroup 4. $n \leq 1500, k \leq 10^6$.

Consider the following algorithm: from position i in the original array, we greedily collect gifts until the end of the current stack. Then we will collect from the next stack, as long as it does not increase the number of subsegments. For each position i in the original array, we will remember the position following the one where we stopped, as well as the number of subsegments obtained in the process, and call these values $go[i]$ and $cnt[i]$.

We calculate the arrays go and cnt for $O(n^2 \log n)$ straightforwardly. Now, to calculate the answer, we introduce a parameter cur, which will indicate the position in the array up to which we optimally divided the array. We iterate from 1 to k, adding $cnt[cur]$ to the answer on each iteration, and changing cur to $go[cur]$. The answer calculation takes $O(k)$, adding the pre-calculation, we get $O(n^2 \log n + k)$.

Subgroup 5. $k \leq 10^6$.

We will learn to find the arrays go and cnt in $O(n)$. To do this, we will use the two-pointer method to find the maximum length of a subsegment containing no more than t numbers for position i, which we will call len. Calculating the arrays go and cnt from the end of the original array, the recalculation for position i can be done as follows:

- If $i + len \geqslant n - 1$, then $go[i] = (i + len)\%n + 1$, where % is the modulo operation, and $cnt[i] = 1$.
- Otherwise, $go[i] = go[i + len + 1]$, where % is the modulo operation, and $cnt[i] = cnt[i + len + 1] + 1$.

Calculating the answer as in subgroup 4, we get a solution for $O(n + k)$ or $O(n \log n + k)$.

Complete Solution.

We optimize the answer calculation. Notice that if $k > n$, then when calculating the answer, the value of cur will take on some values several times. Find the first repetition. Then find the number of steps at which this repetition occurred, as well as the value added to the answer during this time. By dividing the remaining number of stacks by the number of steps, we find how many more repetitions will occur, and process the remaining at the end of the stack separately.

In total, the solution is for $O(n)$ or $O(n \log n)$.