Ostriches

Author: Aidos Nurmash / Editorial: Der_Vlapos

Some important observations:

1. In the optimal solution, the photo segments (contiguous groups of ostriches in each photo) do not overlap, and there is no pair of segments where one is entirely contained within another.
2. Every segment must contain exactly one global maximum height ostrich in the optimal distribution of photos. If it's not true, it is possible to rearrange segments in such a way that each segment has a maximum and the number of required segments won't increase.

Subtask 1: $q = 1$, $m = n$

We need to photograph all ostriches. A greedy approach works:

- Traverse the array from the first ostrich.
- Form the largest possible segment that contains exactly one maximum.
- Once a second maximum would appear in the same segment, close off the current segment and start a new one.

Time complexity: $O(n)$

Subtask 3: Every number x appears at most twice

If each height appears at most twice, let $cntMax$ be the number of ostriches having the global maximum height. Since $cntMax \leq 2$, we need only 1 or 2 photos.

Let $p_1, p_2, \ldots, p_{cntMax}$ be the positions of the maximum elements. Define an array d of length $cntMax+1$ as follows:

- d_1 = number of ostriches before p_1.
- For $2 \leq i \leq cntMax$, d_i = number of ostriches between p_{i-1} and p_i.
- $d_{cntMax+1}$ = number of ostriches after p_{cntMax}.

To check if we can cover m ostriches with 1 photo, we see if there exists an i such that: $d_i + d_{i+1} + 1 \geq m$. If such an i exists, the answer is 1; otherwise, it is 2.

Time complexity: $O(n + q)$

Subtasks 4 and 5: General Case

We compute, for each k, the maximum number of ostriches that can be photographed with at most k photos. Then, to answer a query m, we find the smallest k such that this maximum coverage is at least m (via binary search).

Let's do the opposite; instead of calculating the minimum number of photographs we need to take to photograph at least m ostriches, let's calculate the maximum number of ostriches we can photograph if we take no more than k photographs. Then we can answer queries in $O(\log n)$ time using binary search over this array. This way leads us to write $dp[i][j][f = 0/1]$ = the maximum number of ostriches we can photograph, if we have considered the first i maximums, took j photographs, and if $f = 1$ means we took all ostriches between the i-th and $(i + 1)$ positions of the maximums to the last photo; otherwise, $f = 0$. Then transitions are the following:

$dp[i][j][0] = \max(dp[i-1][j][0/1],\ dp[i-1][j-1][0] + d[i] + 1)$ $dp[i][j][1] = max(dp[i-1][j][0/1],$ $dp[i-1][j-1][0] + d[i] + d[i+1] + 1)$

This can be calculated in n^2.

Time complexity: $O(n^2 + q \log n)$

Full score:

Let's look at how the answer is built.

- Let $c_i = d_i + d_{i+1}$ represent the "gain" from choosing the i-th maximum to form a photo that captures both d_i ostriches to the left and d_{i+1} ostriches to the right of that maximum (counting that maximum itself in the total as well separately).
- We want to select up to k of these c_i-values with the constraint that no two selected indices i and j are adjacent (because we can't intersect segments). Define the set of chosen indices as S.
- The final answer for using k photos is $\sum_{i \in S} c_i \ + \ k$ if $k * 2 \leq cntMax + 1$; otherwise, as $\sum d_i \ + \ k$.

Now let's look at the case when $k = 1$. The answer for this query will be $\max c_i$. Denote that index by i'.

Let's try to enlarge to $k = 2$. The optimal solution will have in S either i' or both $i' - 1$ and $i' + 1$. Because if it has one of $i' - 1$ or $i' + 1$, we can replace it with i', and the answer will not decrease (as $c'_i \geq c_{i'-1}, c_{i'+1}$).

That's why we create a new sequence of costs by removing $c_{i'-1}$, $c_{i'}$, $c_{i'+1}$ and then inserting $c_{i'-1} \ + \ c_{i'+1} \ - \ c_{i'}$ into the same location. Effectively, this corresponds to choosing $i' - 1$ and $i' + 1$ together instead of i'.

Hence, the answer for any k will be the answer for $k - 1$ plus the current (rebuilt) $\max c_i$.

We can implement such local updates using a linked structure (to effectively delete, insert values and handle adjacent elements) and a std::set to manage the biggest values of c_i.

Time complexity: $O(n \log n + q \log n)$