Task: Elephants

Proposed by: Mihai Pătraşcu

1 Subtask 1

In subtask 1, there are only two elephants. Thus we can easily determine the number of required cameras in constant time. Namely, we only need one camera if the elephants are at most distance L apart; otherwise, we need two cameras.

2 Subtasks 2 and 3

If elephants are at positions $x_0, x_1, \ldots, x_{N-1}$, such that $x_i \leq x_{i+1}$ for all $0 \leq i < N-1$, we can compute the minimum number of cameras required using a greedy algorithm. We start with an empty set of cameras. While the current set of cameras do not cover all elephants, we choose an elephant which is not already covered with the minimum position x_j, and place a camera to cover elephants at positions in $[x_j, x_j + L]$. We can implement this procedure to run in time $O(N)$ by iterating through the list of sorted positions once.

Each time an elephant moves in the show, we can update the sorted list of positions in $O(N)$ time. Hence, we have an $O(NM)$ solution to this problem, which is sufficient to fully solve subtask 2, and an adequate optimization is required to fully solve subtask 3. However, a faster algorithm is required for subtasks 4 and 5.

3 Subtasks 4 and 5

3.1 Bucketing elephants

To find the number of cameras, instead of iterating through all elephants, we shall build a data structure that allows us to "jump" over lots of elephants.

We maintain k buckets $B_0, B_1, \ldots, B_{k-1}$ of elephants such that buckets with lower indices store elephants with lower positions, i.e., for any $0 \leq b < k-1$, for all $x_i \in B_b$ and $x_j \in B_{b+1}$, $x_i \leq x_j$. Also, elephants in each bucket are sorted according to their positions.

The goal is to make sure that to find the number of required cameras, one needs to visit each bucket once. For simplicity, we will always place cameras so that the left-most position covered by a camera is the position of some elephant.

Consider bucket b with p elephants. Denote the list of indices of elephants in B_b as $e_0, e_1, \ldots$, e_{p-1} (that is, $x_{e_i} \leq x_{e_{i+1}}$). Given an elephant e_i, we would like to answer the following two questions quickly:

- **Q1:** If we would like to cover all elephants starting from e_i (i.e., elephants in the set $\{e_i, e_{i+1}, \ldots, e_{p-1}\}$), how many cameras are needed?
- **Q2:** What is the highest position that these set of cameras cover?

For elephant e, denote the answer for Q1 for as $J(e)$ and the answer to Q2 as $T(e)$.

If we have these answers for every elephant in every bucket, we can find the number of cameras in time $O(k \log N)$ as follows.

We start by placing the camera at the first elephant in bucket B_0, so that the position of this elephant is the left-most position covered by this camera. Now consider placing a camera at elephant e in bucket B_i in the same fashion. We know that to cover all elephants in B_i, we have to use $J(e)$ cameras and these cameras cover positions up to $T(e)$. We find the first elephant e' not covered by these cameras in the next bucket B_{i+1} by binary searching for the elephant in B_{i+1} whose position is minimum but greater than $T(e)$. Then, we start placing the camera at elephant e' in bucket B_{i+1}.

We repeat this step until we reach the last bucket. Since each step runs in $O(\log N)$ time (from binary search), we spend $O(k \log N)$ time as required.

We can precompute the answers for Q1 and Q2 for elephants in B_i in $O(|B_i|)$ time by iterating over each elephant e_j from e_{p-1} to e_0 and keeping a pointer to the first elephant outside the range $x_{e_j} + L$. For implementation details, please see the appendix.

It is crucial to note that we can process each bucket independent of all other buckets.

3.2 Updating the data structure

When an elephant e moves, we will have to update two buckets: the current bucket B_i and the new bucket B_j. This can be done in time proportional to the current size of the bucket. To find the current bucket of e we can store a pointer from e, but it takes $O(k)$ to find the new bucket anyway. Therefore, the running time for the update is $O(k + |B_i| + |B_j|)$.

Note that the time depends heavily on the size of each bucket. Initially, we would have about N/k elephants in each bucket, but the number may grow as elephants can move. To keep the size of each bucket bounded above by $O(N/k)$, we will rebuild the whole data structure for every $\lceil N/k \rceil$ updates. The rebuilding takes time $O(N)$.

3.3 Choosing the right parameter

We need to handle M updates and answer one question after each of these updates. The total running time is

$$O(M \cdot k \log N) + O(M \cdot (k + N/k)) + O(M \cdot (N/(N/k))),$$

where the first term denotes the total query time, the second term denotes the total updating time, and the last term denotes the total rebuilding time.

Choosing $k = \sqrt{N}$ gives the running time of $O(M\sqrt{N} \log N)$, which is sufficient to obtain full marks for this problem. However, an inefficient implementation may not be able to solve subtask 5. For example, using the set data structure in the C++ Standard Template Library can introduce an extra factor of $\log n$ to the running time of rebuilding the data structure. This can be avoided by using simple arrays.

A Processing each bucket

To simplify the presentation, we add a dummy elephant e_p at position $x_{e_{p-1}} + L + 1$. Also, we let $y_j = x_{e_j}$ be the position of the j-th left-most elephant in bucket B_i.

We consider each elephant j from the right-most elephant e_{p-1} to the left-most one. We also maintain an index t that points to the left-most elephant e_t whose position $y_t > y_j + L$. Initially, $j = p - 1$ and $t = p$.

For each elephant e_j, we will compute $J(e_j)$ and $last(e_j)$, the left-most elephant in the right-most camera in the set of cameras covering $\{e_j, e_{j+1}, \ldots, e_p\}$.

For the dummy node, we let $J(e_p) = 0$ and $last(e_p) = e_p$. For elephant e_j, we check if we need to move t, i.e., if the position of e_{t-1} is greater than $y_j + L$; if that's the case we find the smallest t such that $y_t > y_j + L$. We let $J(e_j) = J(e_t) + 1$ and $last(e_j) = last(e_t)$.

Finally, for each elephant e_j such that $last(e_j)$ points to the dummy elephant e_p, we change $last(e_j)$ to e_j.

We can complete the process using only one pass over all elephants in the bucket B_i and note that the pointer t moves over each elephant only once. Thus, the running time is $O(|B_i|)$ as claimed.

To answer question Q2 for each elephant e_j, we report $y_{last(e_j)} + L$.