Growing Trees

We maintain a sorted sequence of N tree heights in a data structure. The data structure has to support increasing the heights of all trees in a given interval. We use a *segment tree*, which supports the following operations:

- `INC(x, y)` — increase by one all elements in the sequence, whose indices are between x and y,
- `GET(x)` — retrieve the height of the tree, whose index is x.

Both operations run in $O(\log N)$ time.

The simple approach to fertilizing the trees does not work. Consider the following sequence of tree heights:

$$1, 3, 4, 5, 5, 5, 5, 6.$$

Now assume that we fertilize 4 trees of height at least 3. If we simply add 1 to the numbers with indexes 2 to 5 in the sequence:

$$1, \underline{3, 4, 5, 5}, 5, 5, 6,$$

we will get:

$$1, 4, 5, 6, 6, 5, 5, 6.$$

The above sequence is not sorted anymore. To deal with that, we will split the fertilization into two parts. First, we increase all numbers greater or equal to 3 and strictly lower than 5. After that, we increase two trees of height 5, but we choose the last two trees of this height.

$$1, \underline{3, 4}, 5, 5, \underline{5, 5}, 6.$$

As a result, the sequence looks as follows:

$$1, 4, 5, 5, 5, 6, 6, 6.$$

This approach can be generalized easily.

Let us now describe the process in more detail. Assume that we have to fertilize c trees of height at least h. First, we find the first element of the sequence, whose value is at least h. We have the numbers in the sorted order, so we can use binary search for that. Note that since the sequence is maintained in a segment tree, retrieving an element from a given index takes $O(\log N)$ time. This means that the binary search runs in $O(\log^2 N)$ total time.

After that, we can find the height of the highest tree that would be fertilized, denote it by x We fertilize all trees whose height y is at least h and strictly less than x. After that, we also fertilize an appropriate number of trees of height x, but we choose the ones with the greatest indexes. Note that computing the intervals to be fertilized requires some binary searches, similar to the described before.

Similarly, to answer a query about the number of trees of height between a and b, we find the index of the first tree of height $\geq a$ and the last tree of height $\leq b$. Hence, both operations (fertilizing and computing statistics) run in $O(\log^2 N)$ time and the whole algorithm in $O(N + T \log^2 N)$.

Improving the time complexity

The running time is determined by the the running time of binary searches, that find the index of the first element, whose value is, for example, $\geq t$ (for some t). This can be done faster, if for each subtree in the interval tree, we maintain the biggest value in it. As a result, when we want to find the first element of value $\geq t$, we start the search in the root of the segment tree and we can easily check which subtree the element belongs to. If the biggest element in the left subtree is $\geq t$, then we continue the search in the left subtree. Otherwise, we have to search in the right subtree. This reduces the running time of one operation to $O(\log N)$ and the overall time complexity to $O(N + T \log N)$.

Task idea by Mateusz Baranowski and Jakub Łącki, spoiler by Jakub Łącki.