

# Dreaming is not harmful

First, let's make a general remark that will help us in solving all the subtasks. Consider the order in which the vertices will be removed in the original tree, and for simplicity, we will call it the removal order.

Let's consider an arbitrary vertex  $v$  and denote by  $S_v$  the set of vertices that come before  $v$  in the removal order. Now, let's define the optimal vertex for nullification. Note that it is pointless to nullify a vertex on the path from  $v$  to the root, as this can only worsen the position of  $v$  in the removal order. When nullifying a vertex  $c$ , which is not on the path to the root, the position of  $v$  in the removal order will decrease by the number of vertices from  $S_v$  in the subtree of vertex  $c$ .

Thus, we need to find a subtree that does not contain vertex  $v$ , with the maximum number of vertices that come before  $v$  in the removal order.

In the future, when we refer to the sum in the subtree, we will mean the number of vertices from  $S_v$  in that subtree.

- **Subtask 1. No more than two bamboos (10 points)**

In this subtask, the tree consists of no more than two bamboos hanging from the root. In this case, the removal order can be found using the two-pointer method. The optimal vertex for nullification is the root of one of the bamboos. To answer the queries, it is sufficient to traverse the removal order and maintain the sum in both bamboos.

- **Subtask 2. Arbitrary number of bamboos (6 points)**

The solution is similar to subtask 1. To find the removal order, we need to efficiently merge several lists, which can be done using a data structure, such as a priority queue or a set. Now, while traversing the removal order, we will maintain the sum in each bamboo and additionally track two bamboos with the maximum sum. To answer a query, we will choose the best one that does not contain the query vertex.

Now we will learn to simulate the process in the general case. We will maintain a set of the root's children in a data structure. In the next iteration, we will remove the vertex with the maximum value from it and add its children. Depending on the chosen data structure, the asymptotic complexity will be  $O(n \log n)$  or  $O(n^2)$ . A priority queue or a set would be suitable for an efficient implementation.

- **Subtask 3. Any simulation (8 points)**

In this subtask, it is required to write any correct simulation. To answer a query, we will iterate over the vertex for nullification and run the simulation. We will obtain a solution no worse than  $O(n^4)$ .

- **Subtask 4.  $O(n^2 \log n)$  (13 points)**

We will iterate over the vertex for nullification, run the simulation in  $O(n \log n)$ , and update the answer for each vertex with its number in the resulting removal order.

- **Subtask 5.  $O(n \log n + nq)$  (11 points)**

We will go through the removal order and maintain a set of visited vertices. To answer a query, we will calculate the sums in the subtrees and choose the maximum that does not contain the query vertex.

- **Subtask 6. Balanced binary tree (9 points)**

We will go through the removal order and maintain the sum in each subtree. This information can be recalculated in  $O(\log n)$  when considering the next vertex. Note that the optimal vertex for nullification is adjacent to the path from the query vertex to the root. To answer a query, we will consider all such vertices and obtain a solution in  $O(n \log n)$ .

- **Subtask 7.  $O(n \log n + nh)$  (11 points)**

We will act similarly to subtask 6. In addition to the sum in the subtrees, we will also maintain two children with the maximum sum for each vertex. Recalculating such information when adding

a vertex and finding the subtree adjacent to the path to the root with the maximum sum can be done in  $O(h)$ .

- **Subtask 8. Min-heap (14 points)**

It can be noted that in this case, each subtree adjacent to the path from the query vertex to the root either goes entirely to the query vertex in the removal order or entirely after it.

We will take advantage of this and pre-calculate the sizes of the subtrees, then we will traverse the tree in depth. We will maintain the size of the optimal subtree adjacent to the path. When at the next vertex, we will sort its children by value. Children with larger values will entirely go in the removal order before children with smaller values. We will sequentially start a depth-first traversal from them, updating the value of the optimal subtree with the size of the children before them.

As a result of such a traversal, we can also write down the removal order in linear time. We will obtain a solution in  $O(n)$ .

- **Subtask 9. Full solution  $O(n \log^2 n)$  (18 points)**

Similarly to subtask 6, we will go through the removal order and maintain sums in the subtrees. For this, we will use the Heavy-Light Decomposition (HLD) data structure. When considering the next vertex, we will add 1 on the path to the root. To answer a query, we will subtract  $\infty$  on the path to the root (to avoid considering subtrees containing the query vertex), take the maximum in the tree, and then cancel the subtraction.