City of a Thousand and One Windows

Task Author: Ahmet Kaan Avcı

To find the minimum possible maximum work-zone size, we can binary search the answer. For a fixed maximum capacity X, we need to check if we can divide the tree into exactly k_i valid components such that no component has a size strictly greater than X.

Subtask 1: $N \leq 2000$ and $\sum k_i \leq 4000$

For a fixed X, we can check if it is possible using a simple tree DP in $O(N)$ time. We process the tree bottom-up. Each node u will pass exactly one type of component to its parent:

- $dp_0[u]$: The size of the component if it contains **no** crews (empty component).
- $dp_1[u]$: The size of the component if it contains **exactly one** crew.

Let W be the accumulated "empty weight" of node u. Initially, this is 1 (for u itself) plus the sum of all dp_0 values from its children.

We look at the children that pass up a crew component (dp_1). We have to decide whether to cut them or absorb one of them into u:

- **If u is a crew node:** It already has a crew. It cannot absorb another crew from its children. So, we must cut all crew children. Node u forms a crew component of size W. If $W > X$, this X is invalid. Otherwise, u passes $dp_1[u] = W$.
- **If u is an empty node:** It can absorb at most one crew child. To keep sizes as small as possible for higher nodes, we greedily pick the child that gives the smallest merged component. We absorb this child, cut the rest, and pass $dp_1[u]$. If we decide to cut all children (or if there are no crew children), u just passes an empty component $dp_0[u] = W$.

When we cut a child v, we must ensure $dp_1[v] \leq X$. Otherwise, it fails.

Subtasks 2 & 3: Line Graph and $k_i = 2$

In a line graph, there are long paths of empty nodes between crews. When a crew component moves up the path, it absorbs empty nodes one by one, and its size grows monotonically.

To minimize the weight passed to higher nodes, we want to push the cut point as high up the path as possible without the crew component size exceeding X. Because the size is monotonic on a path, we can use Binary Lifting to find this exact cut point in $O(\log N)$ time instead of walking up the path step-by-step.

Subtasks 4, 5 & 6: The Full Solution

Since $\sum k_i \leq 2 \cdot 10^5$, an $O(N)$ check per query is too slow. We must build a **Virtual Tree** (Auxiliary Tree) containing only the k_i crew nodes and their LCAs. This reduces the number of nodes per query to $O(k_i)$.

But an edge (u, v) in the virtual tree is a long path in the original tree. We need a fast way to handle the hidden empty nodes.

1. Precalculating Subtree Sizes
Before processing queries, we run a standard DFS to find 'sz[x]' (subtree size), 'tin[x]', 'tout[x]', and the binary lifting table 'fa[x][i]' for the original tree.

In the virtual tree, suppose u is the parent of v. Let c be the child of u in the original tree that is an ancestor of v. The weight of the entire branch starting from c is 'sz[c]'. We subtract 'sz[c]' from u's total size to easily track what belongs to u and what belongs to its branches.

2. DP on the Virtual Tree
We run the same DP logic as Subtask 1 on the virtual tree. Let W be the empty weight of virtual node u. When a virtual child v passes a component upwards, we have two cases:

- **If v passes an empty component ($dp\_0[v] \neq \infty$):** We just absorb it. We add its weight plus the remaining path weight to W.
- **If v passes a crew component ($dp\_1[v] \neq \infty$):** We want to cut the path between c and v as high as possible. Let's say we cut just above node x. The new size of v's component will be:

  $$dp_1[v] + sz[x] - sz[v]$$

  We need this size to be $\leq X$. This gives us the condition:

  $$sz[x] \leq X + sz[v] - dp_1[v]$$

  Using Binary Lifting, we find the highest node x that satisfies this condition.

  After finding x, the path is split into two parts: 1. The cut-off empty top part of the path has size $eg = sz[c] - sz[x]$. This adds to u's empty weight W. 2. If u decides to absorb v instead of cutting it, the added weight to u would be $mg = dp_1[v] + sz[c] - sz[v]$.

  Just like in Subtask 1, u greedily chooses the child that minimizes the cost difference $(mg - eg)$ to absorb, and cuts the rest at their optimal x points.

3. Final Root Check
After the virtual tree DP finishes at the virtual root, we are not done. There might be empty nodes in the original tree above the virtual root. The virtual root must pass up a crew component $dp_1[root]$. We add the remaining original top nodes to it:

$$dp_1[root] + sz[1] - sz[root]$$

If this final size is $\leq X$, then the capacity X is possible.

Complexity:
Building the virtual tree takes $O(k_i \log N)$. Processing each virtual edge takes $O(\log N)$ for binary lifting. Since we binary search the answer, checking one query takes $O(k_i \log^2 N)$ time. Overall time complexity is $O(N \log N + \sum k_i \log^2 N)$, which easily passes.