Christmas tree

First of all, I'll write some notations for convenience.

deg_v - degree of vertex v

pw_v - length of edge from v to its parent.

We will calculate $dpdown_{v,w}$ - maximal length of christmas tree that has vertex v with branches of length w in its carcass and the "top"of christmas tree (vertex without branches) located somewhere in subtree of v (potentially it can be v). Notice that we do not consider edge to the parent of v as potential branch.

$dpup_{v,w}$ is the same as $dpdown_{v,w}$ but in the opposite direction, i.e., the "highest"vertex of christmas tree is vertex v.

Notice that we are not interested in w if v has no edges that have length w. That means that we only need to iterate at most over deg_v values of w. Thus, iterating over w for every v will take only m operations.

Let's comeback to $dpdown_{v,w}$. Suppose that we calculated *dpdown* for every vertex in subtree of v.

Let's try to iterate over w and try to calculate $dpdown_{v,w}$. First of all $dpdown_{v,0}$ equals to 0. Then, If we have three or more edges with length w then we can take every son as the next vertex in carcass. If we have exactly two edges with length w then we can take every son as the next vertex in carcass except those two.

For this let's maintain three pairs (mx_1, ver_1), (mx_2, ver_2), (mx_3, ver_3).

mx_1 is maximum value of $dpdown_{ver_1,x} + pw_{ver_1}$ over all sons of v such that $x < w$.

mx_2 is maximum value of $dpdown_{ver_2,x} + pw_{ver_2}$ over all sons of v such that $x < w$ and $ver_2 \neq ver_1$.

mx_3 is maximum value of $dpdown_{ver_3,x} + pw_{ver_3}$ over all sons of v such that $x < w$, $ver_3 \neq ver_2$ and $ver_3 \neq ver_1$.

Then, for the first case when we have three or more edges of length w, $dpdown_{v,w}$ equals to mx_1. In other case, we can exclude two unwanted vertices from mx_1, mx_2, mx_3 and take the remaining maximum.

We can sort all $dpdown_{to,x}$ beforehand and process them while iterating over w in ascending order.

$dpup_{v,w}$ can be calculated in almost the same way, we only need to iterate over w in descending order and $x > w$.

This will work in $O(nlog(n))$ because every $dpdown_{v,w}$ will be looked over at most two times - one while calculating $dpdown_{v,w}$ and other when calculating *dpdown* for parent of v.

After calculating both *dpup* and *dpdown* we now need to "merge"them together to cover cases where lca of the carcass is netheir the "top"of the tree nor the "bottom"of the tree.

Let's iterate over v - lca of the carcass - and over w - length of branches of v. Let's say that we have k edges with length w (including edge to the parent). Then, if $k \geq 4$ we can take two edges with length w as branches no matter which vertices we chose as next and previous in carcass. Otherwise, if $k = 3$ we can take any two vertices as next and previous in carcass only if at most one of them has the edge to the v with length w. If $k = 2$ we can take only vertices that has edge to the v with length $\neq w$.

This time we will maintain four pairs $(mxdown_i, verdown_i)$ and four other pairs $(mxup_i, verup_i)$. Find the right $(mxdown_i, verdown_i)$ and $(mxup_i, verup_i)$ for w the same way we did it when calculating *dpdown* and *dpup* althought this time we need four pairs each.

We will iterate over $verup_i$ as the previous vertex in carcass and $verdown_j$ as the next vertex in carcass. If $k - (pw_{verup_i} == w) - (pw_{verdown_j} == w) \geq 2$ update the answer with $mxdown_j + mxup_i$.

We only need four pairs of both because we only care about the case where $k < 4$, thus both *verup* and *verdown* will have at least one vertex that doesn't have an edge to v with length w.