Dynamic diameter

Subtask 1

In the first subtask the limits are small enough to do literally what is written in the problem statement – after each update we run DFS from each vertex to compute all pairs of distances and report the largest one.

Complexity $\mathcal{O}(qn^2)$.

Subtask 2

In the second subtask, we need to compute the diameter in $\mathcal{O}(n)$. There are two ways:

- The first one is a somewhat standard trick. Use DFS from vertex 1 to find the most distant vertex, let's call it u. The diameter is then the distance to the most distant vertex from u, which can be again found using DFS.
- The second approach is using a dynamic programming on a tree, which will be easier to generalise. Root the tree in vertex 1. For each vertex v, compute D_v – the maximum distance to leaf in a subtree of v. This can be done in $\mathcal{O}(n)$ recursively. Then the answer is just maximum (through all vertices u) of D_u and $\max_{v_1 \neq v_2 \text{ are children of } u}(D_{v_1} + D_{v_2} + w_{u,v_1} + w_{u,v_2})$

Complexity $\mathcal{O}(qn)$.

Subtask 3

When the graph is a star, it means that the diameter consists only of two edges, both of which are incident to vertex 1. We can thus simply maintain the weight of each edge in a multiset and output the sum of the two largest values as an answer. Be careful about $n = 2$.

Complexity $\mathcal{O}((q+n)\log n)$

Subtask 4

In the fourth subtask, we modify the DP from the second subtask slightly by introducing a new value S_u to be the diameter of the subtree rooted at u. We can see that

$$S_u = \max\left(D_u, \max_{v \text{ child of } u} S_v, \max_{v_1 \neq v_2 \text{ children of } u} D_{v_1} + D_{v_2} + w_{u,v_1} + w_{u,v_2}\right)$$

and the diameter equals S_1.

Upon changing the weight of the edge $\{u, v\}$, only the values S_x, D_x where x is an ancestor of u need to be updated. As each update can be processed in constant time (the number of immediate children of each vertex is at most 2) and as the tree is balanced, it's depth is $\mathcal{O}(\log n)$.

Overall this needs $\mathcal{O}(n + q\log n)$ time.

Subtask 5

Assume that the diameter always goes through vertex u. This means that the diameter is found by picking two children of vertex u – v_1 and v_2 – such that $D_{v_1} + D_{v_2} + w_{u,v_1} + w_{u,v_2}$ is maximised.

To calculate each value of D_{v_i}, we can use a segment tree that maintains the distances of each vertex to root. When the vertices are ordered using DFS, an update becomes an addition on a range, so any segment tree with lazy propagation is sufficient.

To calculate the maximum of the sum above, we use the approach outlined in subtask 3 – we maintain a multiset of $D_{v_i} + w_{u,v_i}$ and pick the largest two values.

In subtask 5, we can simply assign $u = 1$. This needs $\mathcal{O}(q\log n)$ time.

Subtask 6

In subtask 6 we cannot pick any single vertex through which all the diameters traverse. Nevertheless, we can still pick a vertex u and find the longest path that goes through vertex u to get a lower bound on the answer. Then, we

remove u from the tree, leaving us with some number of smaller trees. This approach can be recursively applied to them.

Performed naively, this leads to a quadratic performance – for example consider a path on n vertices, where we always select a leaf as the root. We need to limit the depth of this recursion in some way.

We do this by always picking the centroid as a root. Recall that centroid is a vertex that when removed splits a tree into a number of subtrees each having at most $n/2$ vertices where n is the size of the original tree. Every tree has 1 or 2 centroids that can be found in linear time using a simple DP.

If we always pick a centroid, then in $\mathcal{O}(\log n)$ iterations we end up with a couple of trees of size 1, for which the answer is 0, so this can be done in $\mathcal{O}(n \log n)$.

Now we simply maintain the segment tree for each centroid. As each edge is in $\mathcal{O}(\log n)$ trees, each update can be performed in $\mathcal{O}(\log^2 n)$. The overall complexity is $\mathcal{O}((n+q)\log^2 n)$.

Subtask 6 – alternative approach

Let's root the tree. Recall the trick from subtask 2: To compute the diameter, we should find the deepest (most distant from root) vertex of the tree – let's call it u – and then find the maximum distance from u. Since we cannot expect anything about u, this problem can also be formulated as just finding the farthest vertex from a given vertex.

Instead of centroid decomposition, we can use heavy-light decomposition. For each vertex v, there is at most one son s connected to v by a heavy edge. Let m_v denote the maximum of depths of all descendants of v which are not descendants of s, inclusive. On each path of the HLD, we should build a segment tree which stores the values $m_v - 2d_v$, where d_v is the depth of v. Also, let's measure these depths from the top of the path containing v instead of from the root.

When looking for the maximum distance from vertex u, let's move to the root and look at all ancestors of u (including u itself). When we take the son s from which we reached v and compute the maximum m of depths of descendants of v which aren't descendants of s (if $v = u$, there are no such vertices because s doesn't exist), then the maximum of lengths of all paths from u which have v as the LCA is $m + d_u - 2d_v$, where the depths are again measured from the top of the path containing v. The maximum distance is the maximum of all these lengths and it's easy to extend the process that computes it to find the vertex with this distance.

Of course, we can't afford to do that by brute force, but HLD helps. If the last edge we traversed in order to reach an ancestor v was a heavy edge, then $m = m_v$. This isn't the case only when we just reached a new heavy path; there are $\mathcal{O}(\log n)$ heavy paths on any path to the root and therefore just $\mathcal{O}(\log n)$ such vertices. When we're moving from u to the root and reach a path p in a vertex v', we need to compute $m + d_u - 2d_{v'}$ for this vertex. Then, we know that we must visit all vertices v above v' on this path, so the maximum of $m_v + d_u - 2d_v$ for all these vertices is a prefix maximum from the segment tree of path p, plus d_u.

How to compute m? With preorder numbering of the vertices, each subtree becomes a segment, so we just need a segment tree that stores depths from the root and m is the maximum from two segments minus the depth of the top of the current path. This segment tree lets us find depths of individual vertices as well.

Now, we just need to handle updates. When we add a to the weight of an edge from u to v (v is a son of u), we add a to the depths from the root of all vertices in the subtree of v. The values of m can only change for u and the ancestors of u which are reached by a light edge. There are only $\mathcal{O}(\log n)$ of them, so we can afford to recompute m for each of them. Recall that we're storing $m - 2d$ in the tree, so we need to also subtract $2a$ from all vertices on the path containing the updated edge, located below that edge (d changes only for these vertices – this is why we aren't using depths from the root). Therefore, the segment trees built over HLD paths need to support operations "update value", "add to segment" and "get maximum from segment".

The time complexity of this approach is $\mathcal{O}(n + q\log^2 n)$, since we're handling $\mathcal{O}(\log n)$ paths per query and doing $\mathcal{O}(1)$ segment tree operations per path.