1 $K = 0$ subtasks

Those subtasks require just finding tree diameter. First one could be solved, for example, with running dfs/bfs from each node and brute-forcing over all pairs (u, v), and picking one with biggest value of $dist(u, v)$. Second subtask is solved with well known algorithm of finding diameter in a tree in $O(n)$ time.

2 More about general solution

Edges with lengths w_i can be seen here as w_i edges of length 1. We can see that the problem remains the same (except now graph size can't fit in memory, but we'll discuss this part later).

So, let's solve the problem for tree with all edge lengths being 1.

From here and further in text a **diameter** will be called any pair of nodes (u, v) for which $dist(u, v)$ is maximum in this tree, and length of diameter will be called it's **length**.

Obviously, tree can have more than one diameter.

We can make an observation: let's fix some diameter in tree; then **all** diameters in the tree will go through the center of fixed diameter. More specifically, center of **each** diameter in the tree is same node or edge.

This can be easily proven by contradiction, suppose centers of two diameters don't coincide, then their endpoints together will form a diameter of greater length, which can't happen.

From now, further in text this point will be refered as **Center of tree**, and edge will be referred as **Middle edge**. If length of diameter is odd, then Center will be any of endpoints of middle edge.

3 Subtask with $K = 1$

Then subtask with $K = 1$ can be solved by finding edge, for which decreasing it's length will decrease the length of diameter of tree. This can be transformed into finding an edge through which all diameters pass.

Let's see what happens if length of diameter is odd? Then all diameters go through middle edge (all diameters in tree will contain both endpoints of this edge), so decreasing it's length will decrease tree's diameter.

What if it's even? Then we can decrease length of any edge incident to tree's center that belongs to any diameter. Let's see why it works:

Let's colour edges through which at least one diameter goes. If there are at least 3 coloured edges incident to tree Center, then removing any won't decrease diameter. If there are two such, then removing any will decrease diameter. (note that there can never be just one such if length is even).

This subtask may have many other solutions, for example, calculating for each edge how many diameters go through it (Can be done with dfs in $O(n)$), or others.

4 Solution for bigger K

At this moment contestants should make some observations. Next I will describe intended solution and intuition to it, then I will describe alternative approaches.

Let's, firstly, solve an easier and seemingly unrelated problem:

*

Given a tree roted in vertex R, you are allowed to do next operation at most K times, minimize tree's depth.

1. Pick an edge in the tree that has non-zero length.
2. Decrease this edge's length by 1.

Here we define $depth(u)$ with tree rooted in R as $max(dist(u, v))$ over all vertices v in u's subtree, and depth of tree is equal to $depth(R)$.

*

Solution to this sub-problem is very short and simple.

Firstly, it is obvious that all operations are **commutative** so their order doesn't matter.

We can see that applying operation on edge (u, v) (wlog u is parent of v) will decrease value of $dist(root, x)$ for each node x in v's subtree.

Let's also call an edge A ancestor of edge B if A lies on path from B to root.

Let's define *depth* of an edge E connecting nodes u and v as $depth(v) + 1$ if u is parent of v, and $depth(u) + 1$ otherwise.

Let's take a look at operation order: say we have two edges A and B.

We can have two cases:

1) **One edge is ancestor of another:** (wlog A is ancestor of B) Then applying operation on A before B will never lead to a worse result, since it will affect value of $dist(root, u)$ for more nodes, including all nodes that B would affect.

2) **Neither of edges is ancestor of another:** Then applying operation on one edge will **never** affect depth of another one, so applying it on edge with greater *depth* will lead to at least not worse result.

Now we can observe that (1) also satisfies (2), because if A is ancestor of B then $depth(A) > depth(B)$.

This gives us simple **optimal** order of doing operations:

1. Sort edges in non-increasing order of their depths.
2. Apply operation on first K of them.

Let's relate it to original problem.

First of all let's rephrase our operation: we pick and edge E connecting nodes u and v, and we "collapse" the edge (we merge nodes u and v into one new node).

Obviously there exists some optimal set of operations for given tree and K. Let's look at Tree Center **after applying all operations in optimal way**. (Let's call this node O).

Claim: If we root initial tree in O, then the sequence that minimizes it's depth will also minimize it's diameter.

Proof: First of all, for tree rooted in it's Center, length of diameter will be bounded by $2 \cdot depth - 1 \leq length \leq 2 \cdot depth$, where $depth$ is depth of the tree; Also there is more general bound we'll use: For any rooted tree $diameter \leq 2 \cdot depth$; Let's look at $sequence1$ of operations that minimizes diameter and $sequence2$ that minimizes depth:

Lemma: $depth2$ is equal to $depth1$. Suppose $depth2 < depth1$. Then we have simple inequality $diameter2 \leq 2 \cdot depth2 < 2 \cdot depth1 - 1 \leq diameter1$. Which results into $diameter2 < diameter1$ which can't be true because sequence 1 minimizes diameter.

So we can see that apart from minimizing diameter, sequence 1 also minimizes tree's depth.

Next, let's see another inequality: $2 \cdot depth1 - 1 \leq diameter1 \leq diameter2 \leq 2 \cdot depth2$;

Since $depth1$ and $depth2$ are equal, we can show that O will be Center of tree after operations from sequence 2 as well. **Proof:** Note that from inequality above $diameter2$ can be either $2 \cdot depth - 1$ or $2 \cdot depth$. If $diameter2 = 2 \cdot depth$ then it is trivial to see that O is Center. If $diameter2 = 2 \cdot depth - 1$ we can show that O is Center.

A) Suppose diameter doesn't go through O. Then diameter will belong to one of O's children's subtrees. For one of endpoints of diameter $dist(O, U) = depth$, and another endpoint belongs to same subtree. Since diameter doesn't go through O, biggest length of $diameter$ we can have is $2 \cdot (depth - 1)$ (otherwise depth would be greater) which is smaller than $2 \cdot depth - 1$.

B) If diameter goes through O, then since for one of endpoints $dist(O, U) = depth$, there exists other endpoint for which $dist(O, V) = depth - 1$, so O is Center.

Let's look at all edges adjacent to O after applying all operations from sequence 2. Since O is Center, diameter length will be equal to sum of depths of two edges with biggest value of $depth$ among ones adjacent to O. Let Q be sorted sequence of edges generated by minimizing depth algorithm, then we can see that diameter length is equal to $depth(Q_{k+1}) + depth(Q_{k+2})$ (even if edge Q_{k+2} isn't adjacent to Center, there will exist other edge with same depth that will be adjacent to Center).

Let's finally show that $sequence2$ minimizes tree diameter. Let's root tree in O and apply operations of $sequence1$. Remembering that O becomes Center, diameter length will be equal to sum of depths of 2 edges with biggest value of $depth$. So, if we calculate $depth$ of each edge, we can see that diameter length can never become smaller than sum $K + 1$-th and $K + 2$-th greatest depths. Thus $sequence2$ produces smallest diameter we can achieve, which is exactly the answer to problem.

***

So, now we know that there surely exists such node, such that if we root tree in it and minimize it's depth, it will minimize tree diameter as well. Moreover, where Q is sequence of edges generated by minimizing depth algorithm, answer to problem will be $depth(Q_{k+1}) + depth(Q_{k+2})$ then. (assume $depth(Q_n)$ and $depth(Q_{n+1})$ both are 0)

We can try each node in tree as root and take minimum value of $depth(Q_{k+1}) + depth(Q_{k+2})$ among all roots. This will give us correct value because:

1) **If after K operations u becomes Center:** then current diameter simply is $depth(Q_{k+1}) + depth(Q_{k+2})$. All nodes other than O can't give better answer since O becomes Center in optimal choice of operations.

2) **If after K operations u doesn't become Center:** We can see that Q_{k+1} will be parent edge of Q_{k+2}, so actual diameter of tree will be $< depth(Q_{k+1}) + depth(Q_{k+2})$, which means it'll always give greater result than the optimal one.

So among all roots, we should take just minimum value of $depth(Q_{k+1}) + depth(Q_{k+2})$ since among ones that satisfy (1) there will be O so eventually we'll find optimal diameter, and among ones that satisfy (2), value we'll find will be greater than optimal diameter.

So, naive solution we have now - fix each possible root, calculate $depth(E)$ for each edge in O(n), find Q in O($nlogn$), take smallest sum of $depth(Q_{k+1}) + depth(Q_{k+2})$ among all roots. Time complexity: O($n^2 \cdot \log(n)$).

5 O($m \cdot \log(m)$) solution

Let m be sum of lengths of all edges in initial tree. $m = n-1$ for $w_i = 1$ subtasks, and can be greater for weighted tree subtasks. Formally, $m = \sum w_i$ for each $1 \leq i \leq n-1$.

***

It would be great if we could find that optimal root somehow. Let's take a closer look at Q and how exactly it changes if we reroot tree.

Let tree be rooted now in U and there exists edge $U - V$. If we reroot tree from U in V, for which edges does value of their *depth* change? It turns out that while rerooting the tree, only depth of the edge $U - V$ changes, and all other edges' depths remain same.

So, first way to solve problem in O($m \cdot \log(m)$) is: fix arbitrary root, calculate depths for each edge. Then we can apply rerooting technique and try to reroot tree to each vertex possible, while supporting updating depth of edge we just crossed. For each edge we can precalculate it's depth with tree being rooted in each of it's endpoints in O(n) with a simple dfs. After updating depth, simply find $k+1$-th and $k+2$-th greatest values, and update the answer (if needed). This part could be done with a Segment Tree data structure.

Problem of this approach is that it depends on m and not n, so (with getting first 3 subtasks) it would give contestant 70, or even 80 points if they would transform each edge into w_i edges of length 1, but not 100 points.

It turns out optimal root can be found without rerooting and without even searching for it in code. Let's root our tree in it's Center and calculate all depths of edges. Let's see what happens now, if we reroot the tree "further" from center.

*

For shortness, let's call length of diameter D.

Suppose tree is rooted now in U and we reroot it to V, and U is closer to Center than V. We can see that $depth(Center)$ (with tree rooted in U) is at least $D/2$ (since it is **Center**). The same way, $depth(V)$ (with tree rooted in U) is not greater than $D/2$. So, if we reroot the tree, old value of $depth(edge(U-V))$ was less than or equal to $D/2+1$. But new value becomes equal to $depth(U)+1$ (since V is parent of U now). Since tree is rooted in V and U lies on path from $Center$ to V, then U is ancestor of $Center$, so $depth(U) \geq depth(Center)$ (equality happens when U is Center itself).

So we have $depth_{new}(U) \geq D/2 \geq depth_{old}(V)$ which means that $depth(edge(U-V))$ never decreases when we reroot tree further from center, which means sum of $K+1$-th and $K+2$-th greatest values of edges never decreases too. Thus we reach smallest sum of $depth(Q_{k+1}) + depth(Q_{k+2})$ when tree is rooted in **it's Center**.

So our algorithm simplifies to: Finding tree's Center, rooting tree in it, finding sequence Q, taking $depth(Q_{k+1}) + depth(Q_{k+2})$.

This still works in $\mathrm{O}(m \cdot \log(m))$, but already is way simpler.

6 What if length of edge can be greater than 1?

As it was said before, we can see edge of length w_i as w_i **edges of length** 1. Problem is that if we build that explicitly, tree can have up to $2 \cdot 10^9$ edges, which is, of course, too much. So we'll store them normally, but imagine them as set of w_i edges of length 1.

Let's see how exactly does edge $U-V$ of length w contributes to Q? Wlog U is parent of V. Then w edges of length 1 will have their depths equal to: $depth(V)+1$, $depth(V)+2$, ... , $depth(V)+w$.

So basically, one such edge $U-V$ with length w and U being parent of V adds to Q 1 instance of $depth(V)+i$ for each $1 \leq i \leq w$.

In the end we will still have to find $K+1$-th and $K+2$-th greatest values. Does that remind you of anything? What about Segment Tree with range adding and finding K-th smallest/greatest element, which is pretty simple to implement?

So, full solution to problem does the next:

1) Find tree Center. (note!) If there doesn't exist such node that would be *Middle node of a diameter*, artificially create it (by adding a new node on middle edge).

2) Calculate depths of all nodes with dfs.

3) For each edge $U-V$ (U is parent of V) add 1 on range $[depth(V)+1, depth(V)+w]$ in segment tree.

4) Output sum of $K+1$-th and $K+2$-th greatest values in segment tree.

Final time complexity: $\mathrm{O}(n \cdot \log(m))$.

7 Alternative approaches.

There exist some other approaches for this problem, let's discuss them in order.

*

- Guessing optimal strategy (unoptimal time complexity): When I just came up with this problem I had no ideas of any polynomial solution for it. (Initially problem had all weights == 1). Then I made assumption that it's always optimal to apply operation on edge **which belongs to biggest number of diameters** i.e. edge **through which most diameters go**. Sadly I failed to prove it, yet I later stress tested it against all possible solutions I had and it showed that it's correct.

This observation is pretty intuitive, so some participants can come up with it during contest, having no other ideas of improving solution.

Implementing it, however, will give them not more than 40 points, because the most optimal way this can be done is $O(K \cdot N^2)$, by doing K iterations of next algorithm: go through each edge, run dfs from both it's endpoints, then number of diameters can be found in $O(n)$ with 2-pointers or other ways.

*

- Extension of previous solution:

When I came up with this, I thought it will be the full solution. Yet, I optimized it later, but some of participants can still come up with this:

If one observes that all diameters in tree go through it's Center, then edge belonging to most diameters will be incident to the Center.

Thus we can improve previous solution to $O(K \cdot N)$ one: do K iterations of next algorithm - find Center ($O(n)$); root tree in it and find all endpoints of diameters with a dfs in $O(n)$; for each edge incident to center finding number of diameters going through it simplifies to number of endpoints inside subtree multiplied by number of endpoints outside the subtree (in case of even diameter length), for odd length just one edge will be relevant.

There exists other solution with same complexity, which has more difficult implementation but doesn't require tree Center observation. Though, still, this solution is based on observation that there exists **at least one** node in which all diameters in tree intersect (not necessary just Center). It can also be proven by induction, keeping the "intersection" range of diameters and adding new ones 1 by 1, it can be shown that intersection is never empty.

If one can find this intersection point, then solution is same as above one: root tree in this node, calculate number of diameters going through each edge incident to root, except calculating this number won't be as simple as in solution above, still it's doable in $O(n)$.

Either of those solutions could give contestant up to 55 points.

*

- Next step of optimization of this greedy idea: Extending solution with rooting tree in Center, one could observe the next property: Let's (imaginarily) colour all edges that belong to at least one diameter. Firstly, if diameter length is odd, then all diameters will go through middle edge, so in this case we can always remove just it. Now we'll always deal with even length diameter.

Let S be set of coloured edges incident to tree Center at some moment of time. It will always be optimal to remove any edge from this set. Let's show why this works:

1) while size of S is greater than 2, diameter won't change after operation. It's easy to see why.

2) When size of S is equal to 2, removing any of those 2 edges will decrease diameter by 1.

3) When size of S is equal to 1, diameter length becomes odd (see why?), so we're "forced" to remove that edge.

When S becomes empty, recalculate it again.

By just simulating the process we can get simple $O(n^2)$ solution. It can be further optimized either maintaining priority queue of edges, or by calculating *depth* for each edge and sorting list of edges, then applying operation on first K (see why this works?) of them.

So we end up with another, though not proven, $O(n \cdot \log(n))$ solution, which works for unweighted (i.e. $w_i = 1$) tree.

*

- Another way to go from 70 to 100 points: Instead of using segment tree, one could do binary search on biggest depth left in tree, though this one may be tricky to implement, but works as well and doesn't require additional data structure for solution. I want to thank Nemanja Majski for pointing this idea.

*

- Way to solve problem in $O(K \log K)$ without data structures: It requires all observations needed for full solution, but instead of using segment tree one can process operations explicitly, one by one in Dijkstra-like way with priority queue. Possible there even exists way to do this in $O(N \log K)$ but I'm not aware of it.

*

In the very end, I want to thank Alin Popescu and Nemanja Majski for their help with process of proving, improving, and developing solution, and testing the problem.