Journey Scheduling

SK Editorial by Sergey Kulik

First of all, there is a following way of finding the longest distance in the tree, considering that all the edges' weights are non-negative:

- Take an arbitrary node of the tree. Let's say this node's index is A.
- Take any node that is the most distant from A. Let's say this node's index is B.
- Find the most distant node from B. Similarly, let's call this node's index C.
- Now, there is a follwing claim: the distance between the node B and the node C equals to the largest distance between any two nodes in the given tree.

Let us now consider a particular query:

- First we move from the given node A to the most distant node B.
- Then we move from B to the most distant node C. That means that on this step we've passed the distance equal to the largest distance between two nodes in the tree.
- Then we move from C to the most distance node D. But we can take B as A, C as B and D as C and that means that again we pass the distance, equal to the largest distance between two nodes in the given tree.
- And so on...

So now it is clear the the answer is $(K-1)$ times the largest distance between any two nodes plus the largest distance from the particular query node. This distance can be calculated using the standard tree DP approach (described below).

Consider any arbitrary node in the tree call it u. Now, all paths from u either go through the parent of u, or go down from u and contain at-least one of u's children. Denote $down[v]$ as the maximum length of the path which starts at v and goes down (away from root). Similarly, $up[v]$ denotes the maximum length of the path which goes up from v (towards the root). Now, $farthestDistance[u] = maximum(all\ down[v] + 1\ such\ that\ v\ is\ u's\ child,$
$up[p] + 1\ where\ p\ is\ u's\ parent,\ down[v]\ + 2 such\ that\ v\ is\ a\ child\ of\ p\ and\ v! = u)$.

Now, the first 2 quantities can be calculated through a single dfs.

The third one needs a little trick to do in time.

For every node u, apart from having the length of just the longest path that goes down, store the length of the second longest path also.

Hence, $max(down[v] + 2\ where\ v\ is\ a\ child\ of\ p\ and\ v \neq u)$ becomes $max_1\_down[p] + 1\ if\ this\ does\ not\ correspond\ to\ u,\ max_2\_down[p] + 1$ otherwise.

This is an example of standard tree dp. Its DP since you use the stored information of other nodes to get your answer.

Feedback

Was this editorial helpful?

Yes No