Trees 2

DFS Traversal Application

- Find the level of each node
- Find the parent of each node
- Find the number of children of each node
- Find the subtree size of each node
- Find the farthest leaf node from each node in subtree
- Find the longest path passing through each node as LCA

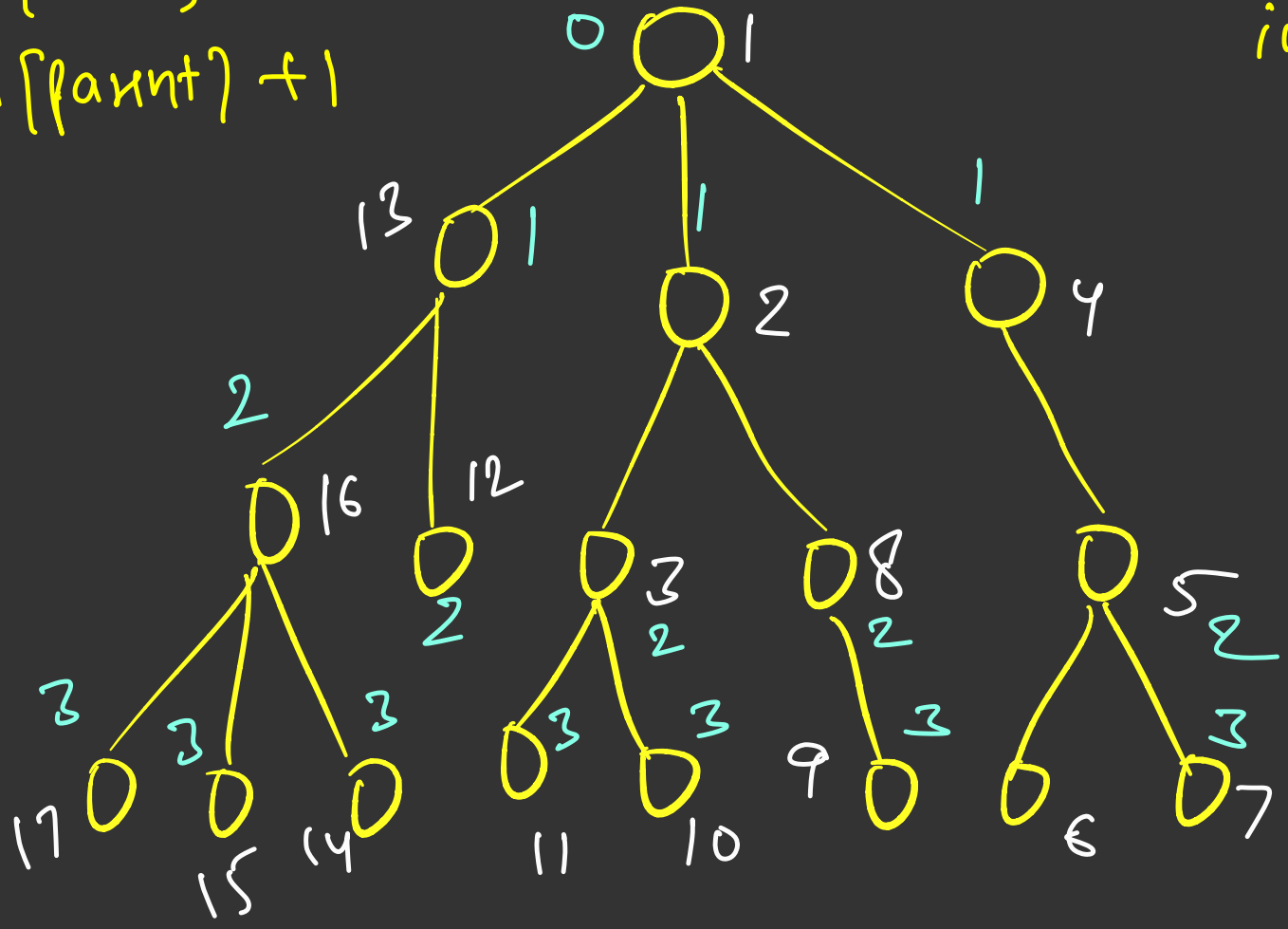
use the parent parameter being passed

distance of the farthest leaf node

Problem ①

② level[node] = level[parent] + 1

① pass a parameter in DFS



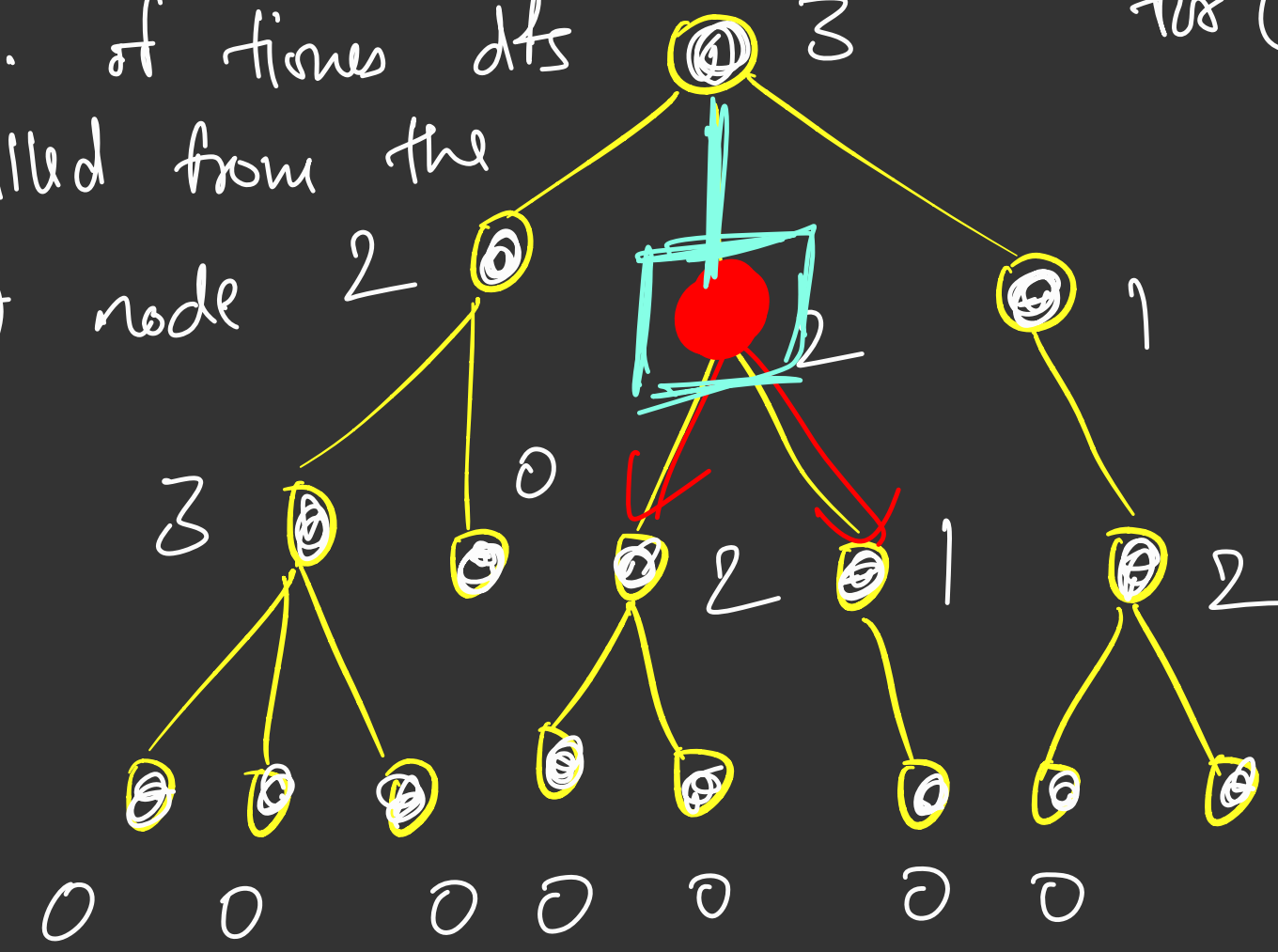
Problem (3)

dfs (node)

for (neighbour)

if (n == p) continue

no. of children for a node = no. of times dfs gets called from the current node

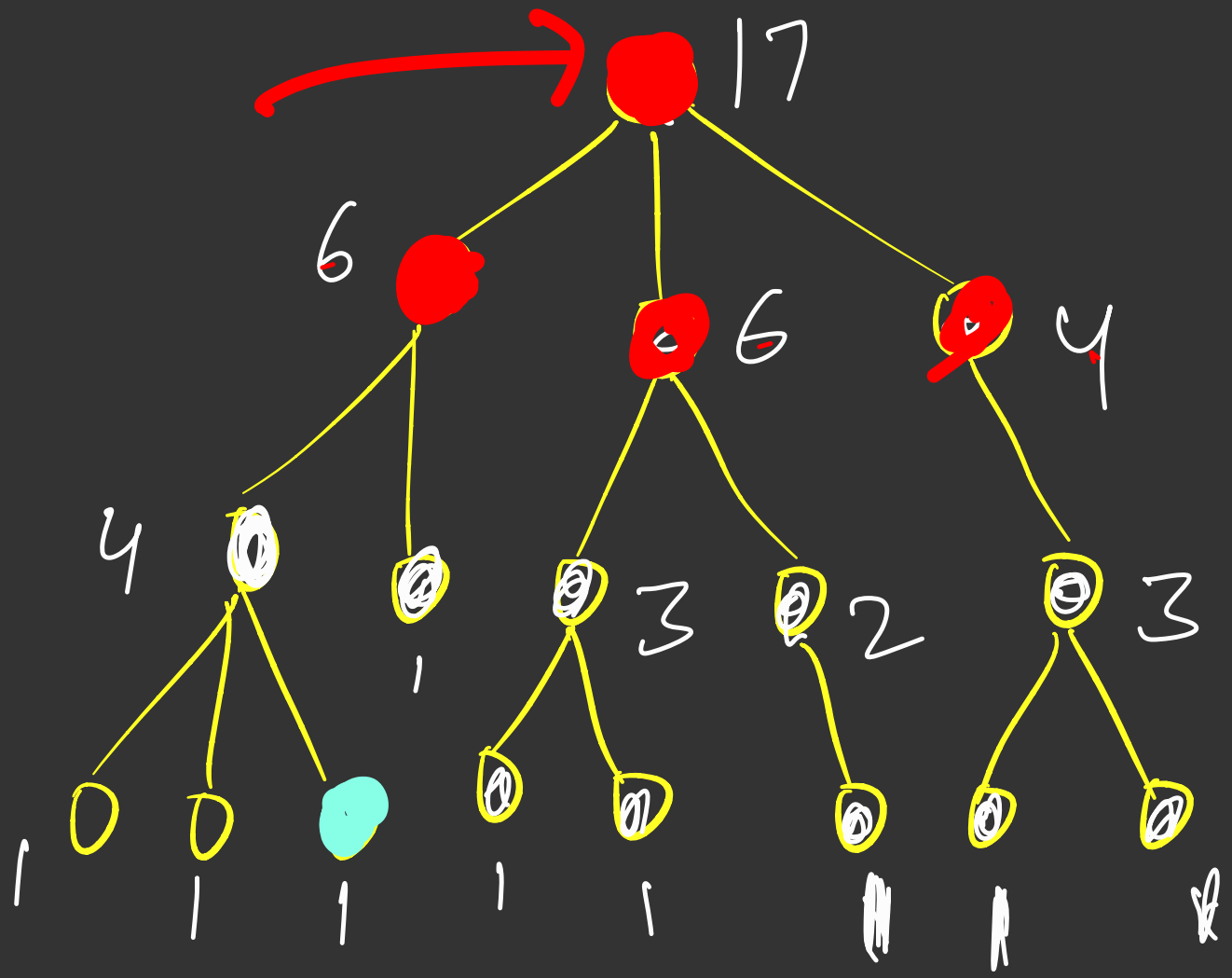


size of edges[i] = no. of neighbours of i

if node == root, children = edges[node].size();

else, children = edges[node].size() - 1;

IncSum ④

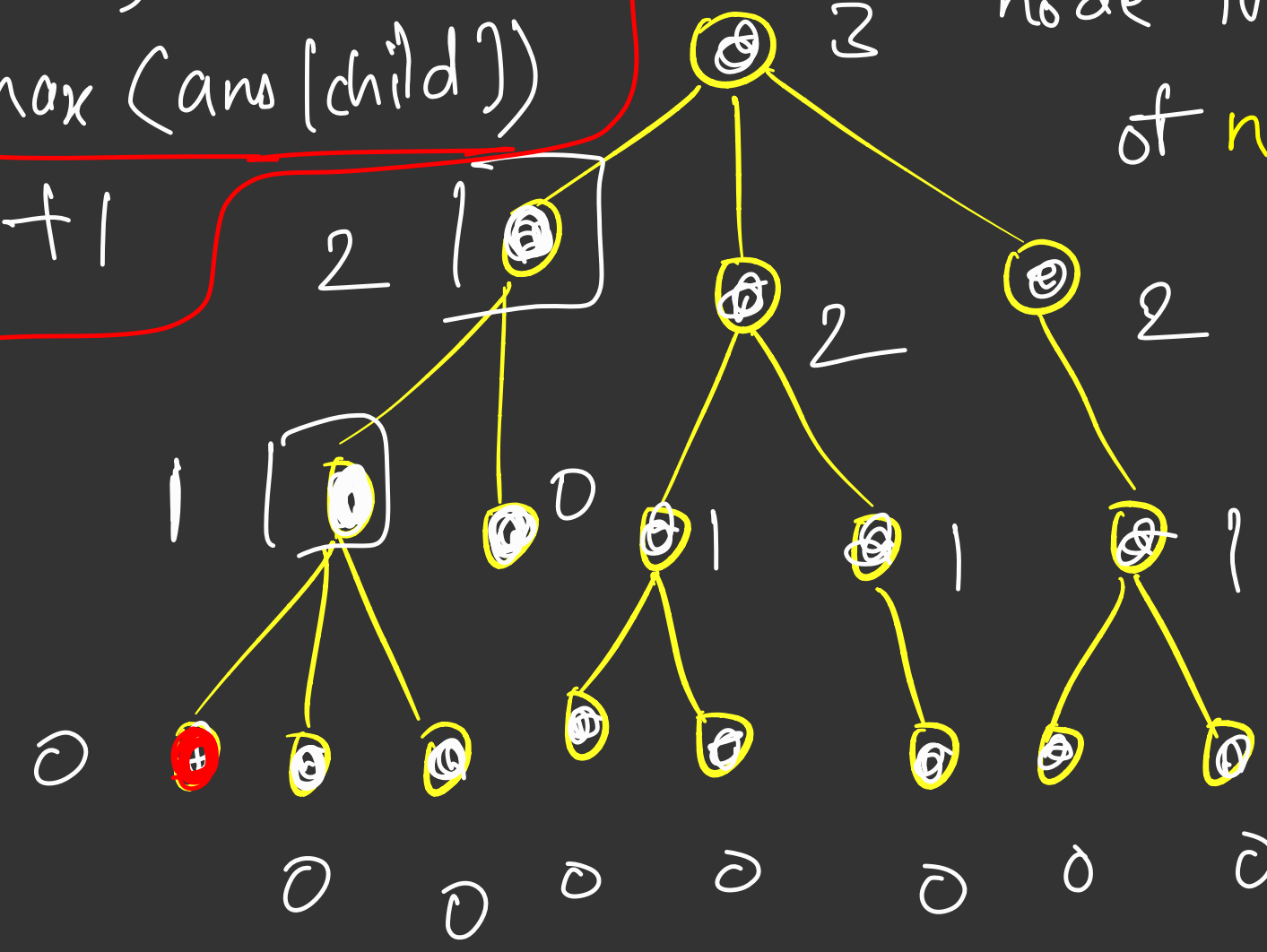


$$sub(node) = \left(\sum_{x \in children(node)} sub[x] \right) + 1$$

Problem (5)

ans(node) = max level of any node in subtree of node − level[node]

ans(node) = max(ans(child)) + 1



max_level(node) = max(max_level(child))

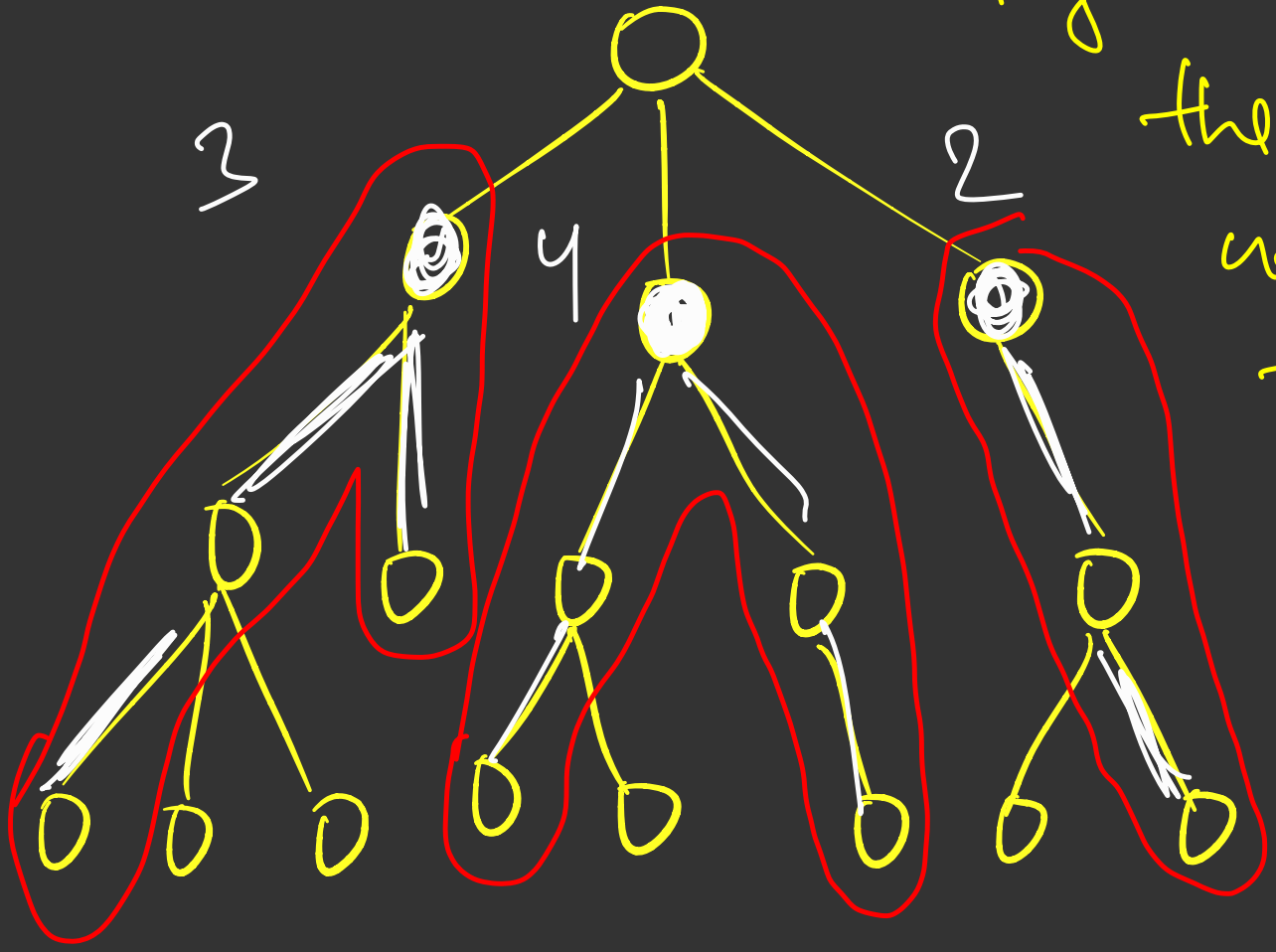
① $ans[node] = max(ans[child]) + 1$

② $ans[node] = max\_level[node] - level[node]$

$max\_level[node] = max(level[node], max(max\_level[child]))$

Problem (6)

for each node find out the longest path with the current node as the top node

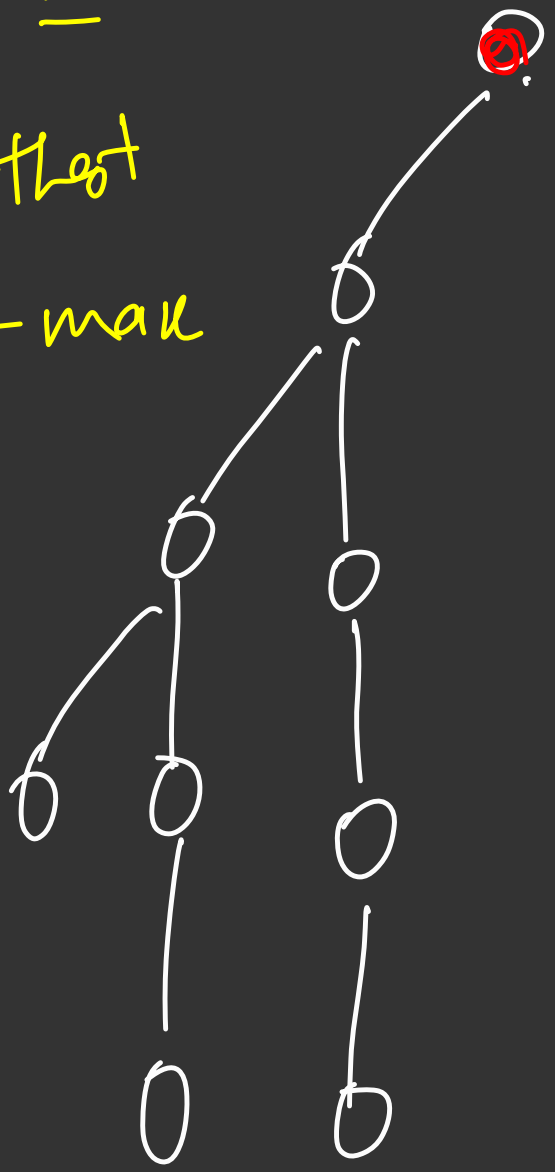


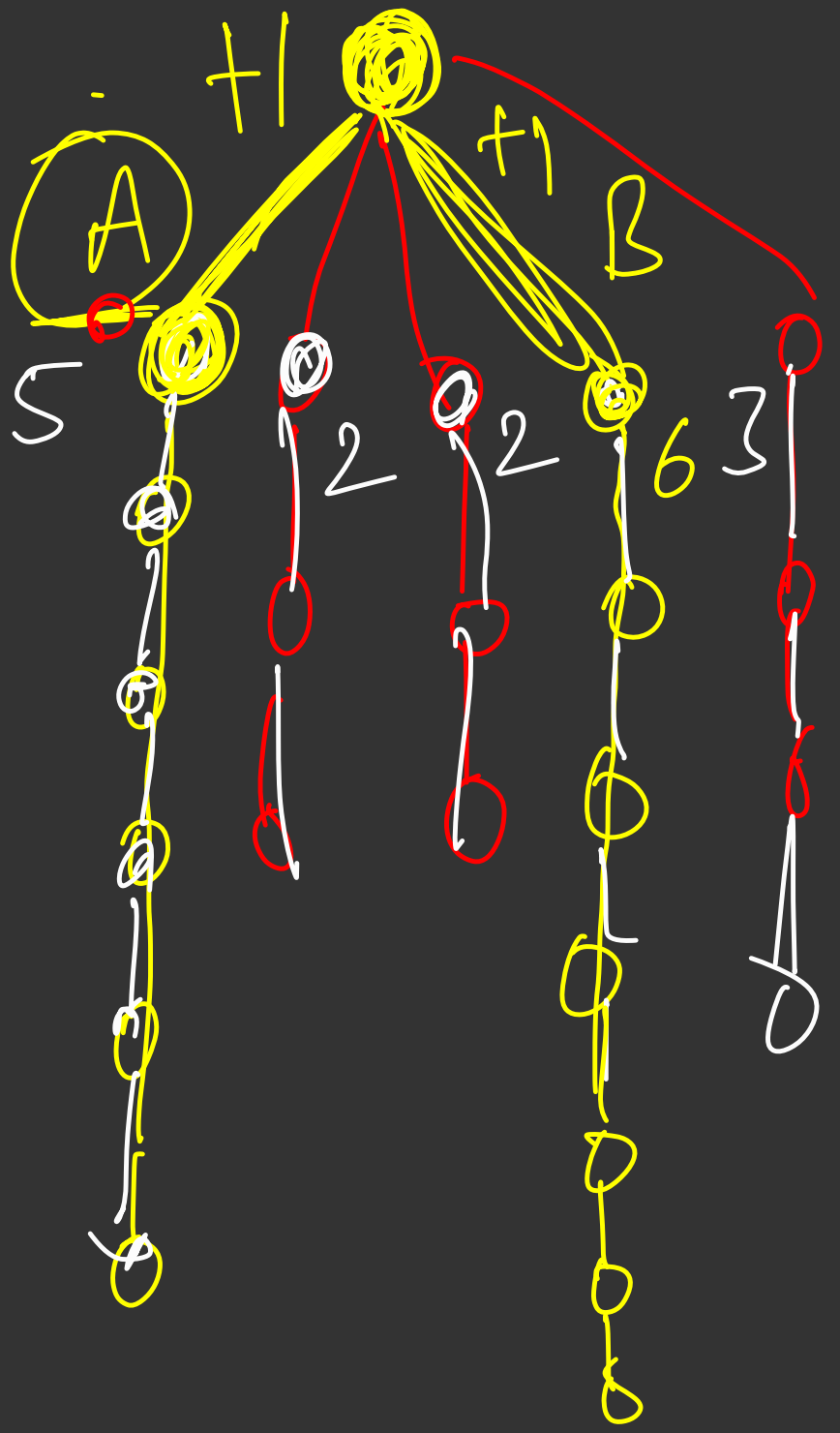
ans(node) =

max_farthest

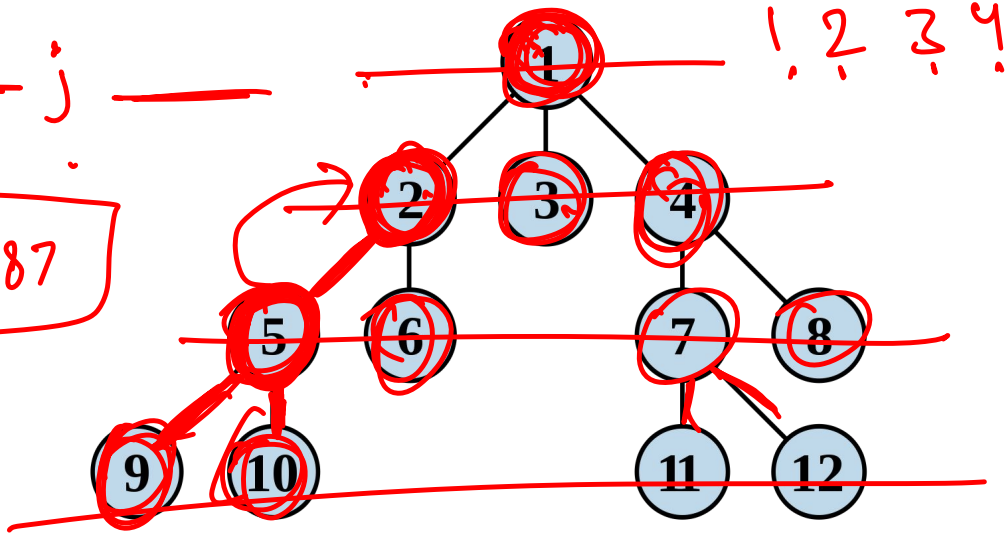
+ secnd_max

+2





BFS Traversal in a Tree

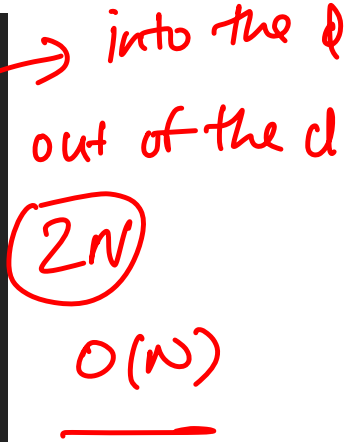


Nodes are numbered in the order in which they are visited

BFS Traversal in a Tree

Implementation:

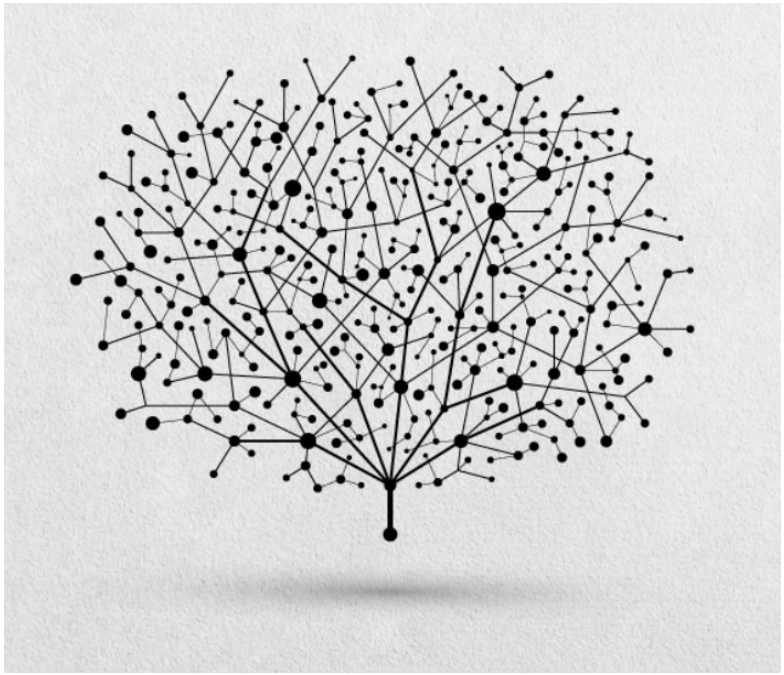
```
void solve(){
    int n;
    vector<vector<int>> adj(n);
    for(int i = 0; i < n - 1; i++){
        int u, v;
        cin >> u >> v;
        u--, v--;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    int root = 0;
    vector<int> bfs_traversal;
    queue<int> qu;
    vector<bool> visited(n, false);
    qu.push(root);
    visited[root] = true;
    while(!(qu.empty())){
        int currentNode = qu.front();
        qu.pop();
        bfs_traversal.push_back(currentNode);
        for(int neighbour : adj[currentNode]){
            if(!visited[neighbour]){
                visited[neighbour] = true;
                qu.push(neighbour);
            }
        }
    }
}
```



Time Complexity: O(N)

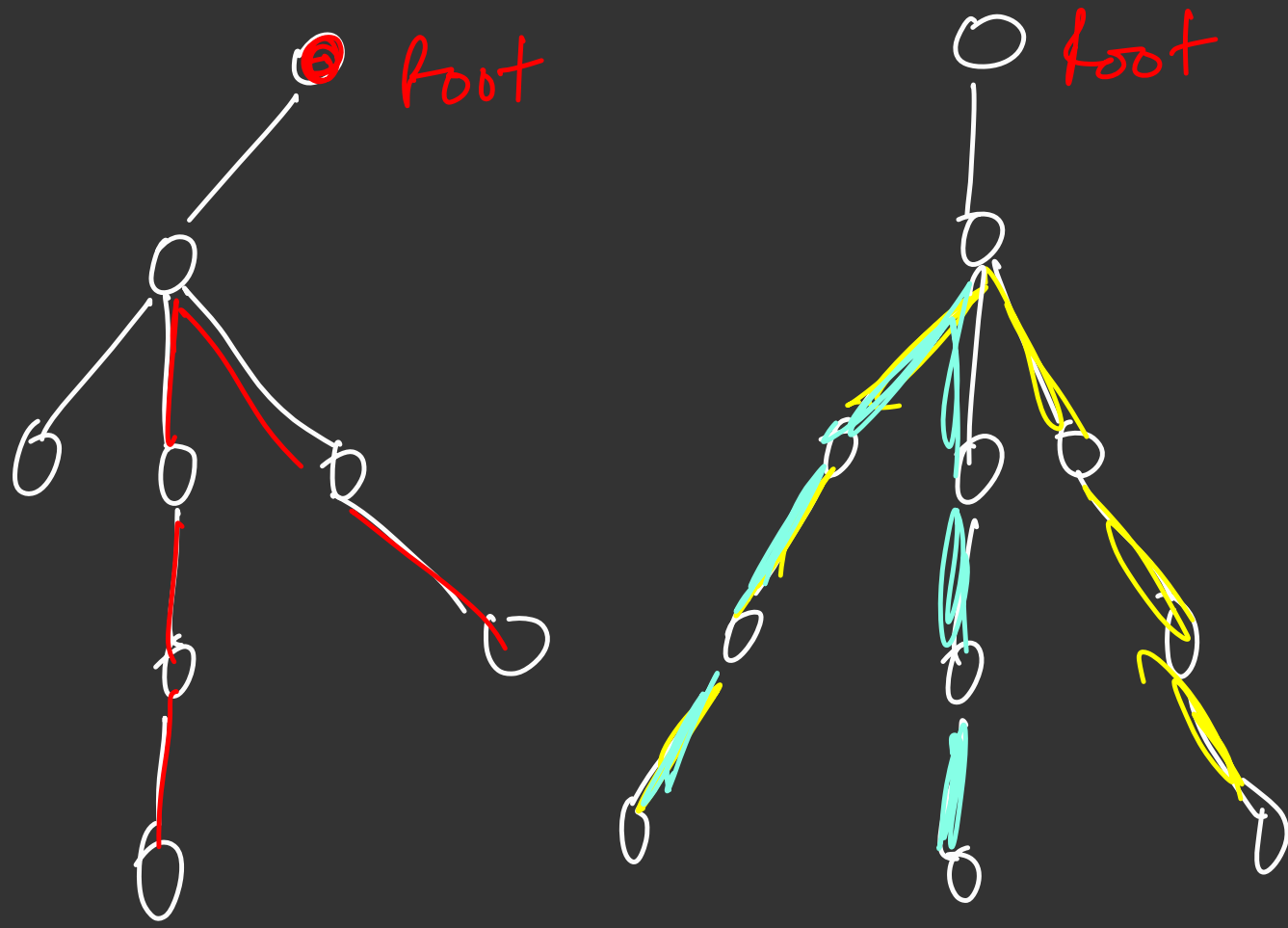
Diameter of a Tree

Given a Tree with N nodes find out its diameter O(N)



Diameter of a tree = Maximum distance between any 2 nodes in the tree

Problem: Link

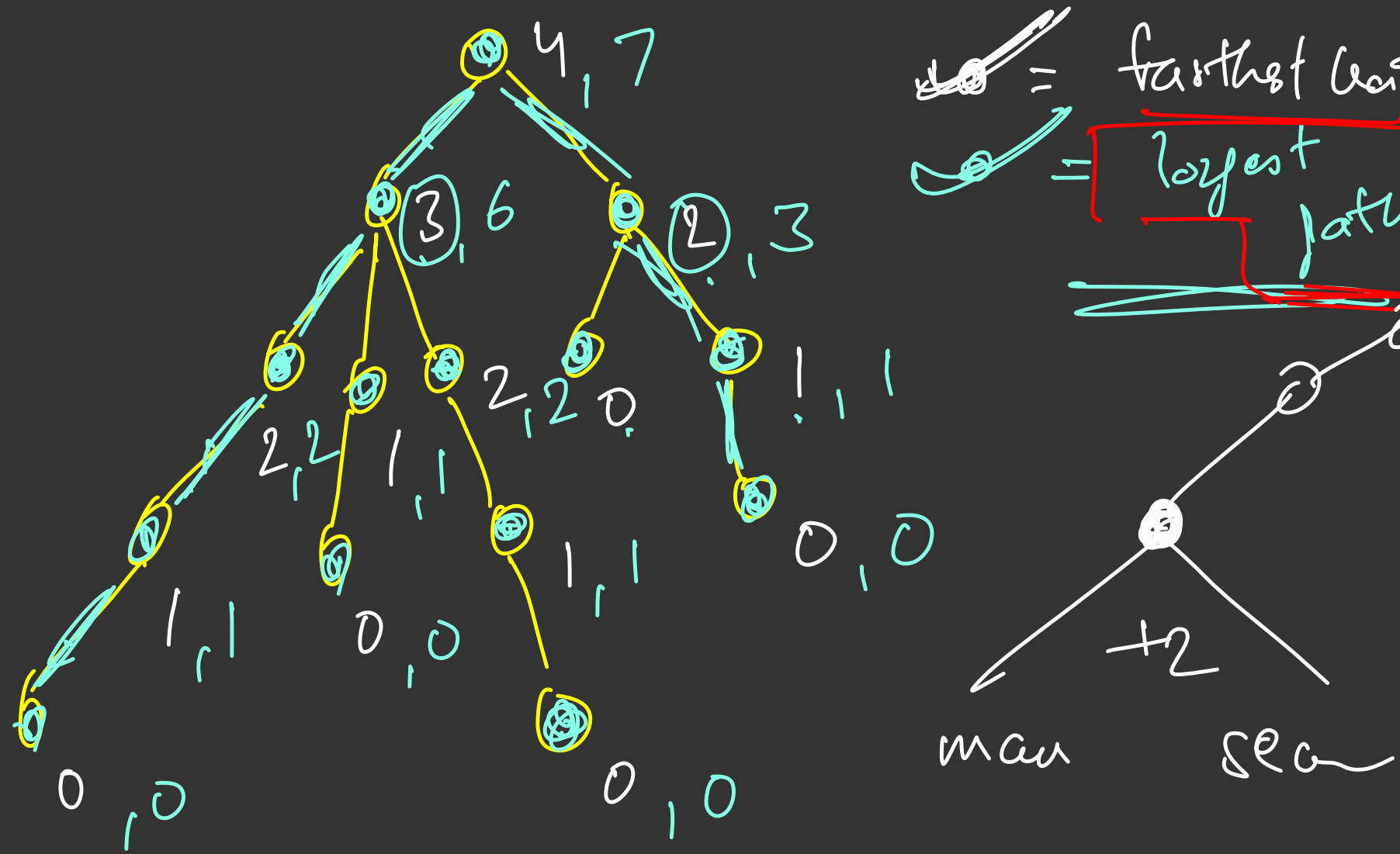

root
root

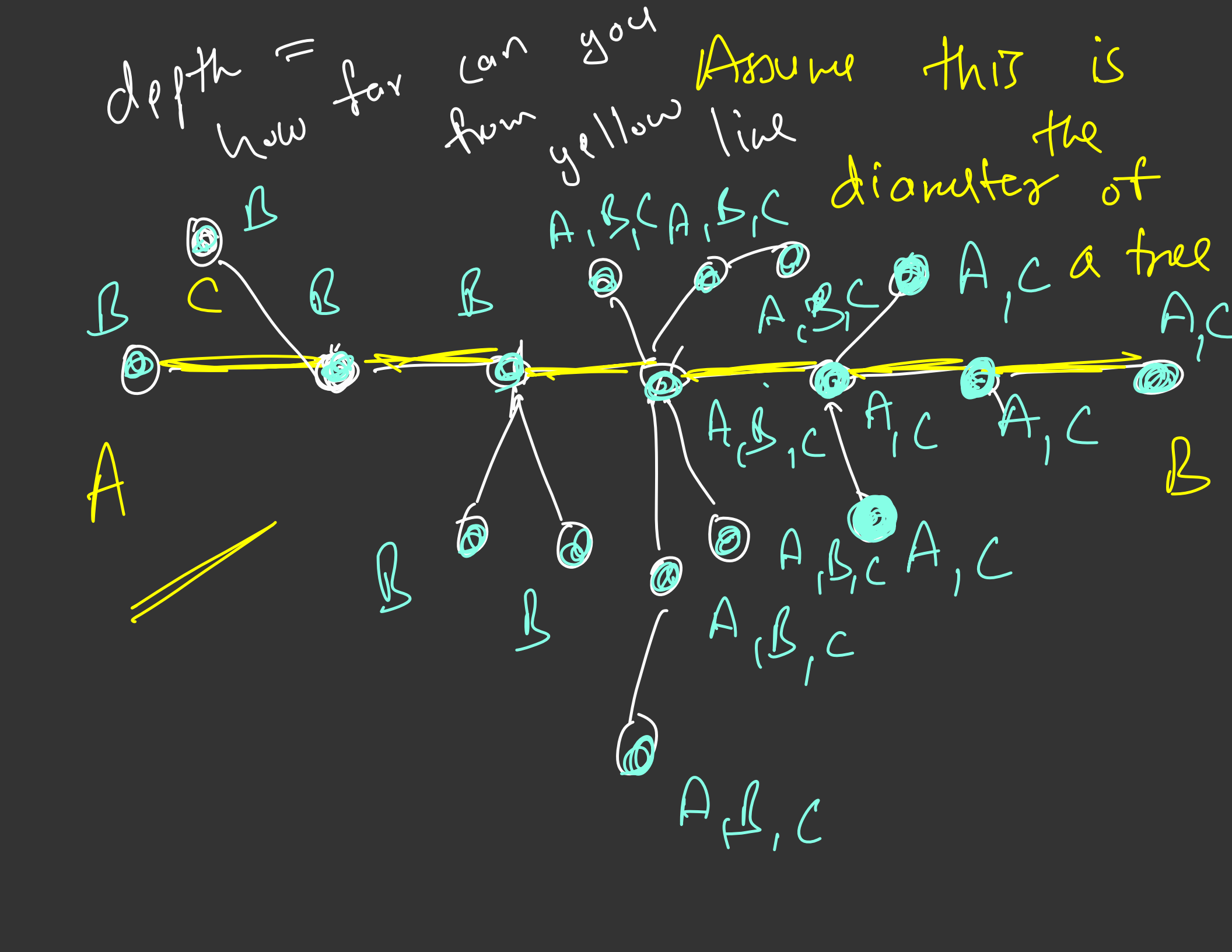
Problem 6 :

for each node find out the longest path with that node being the top node and call it ans[node]

Diameter of tree = max(ans[node])

max(ans[node]) = maximum of the longest paths with a top node fixed.


4,7
3,6
2,3
2,2
1,1
2,2
0,
1,1
1,1
0,0
1,1
0,0
0,0
0,0
= farthest leaf
= longest path
+2
max
sec


depth = how far can you from yellow line
Assume this is the diameter of a tree
A
B
C
A,B,C

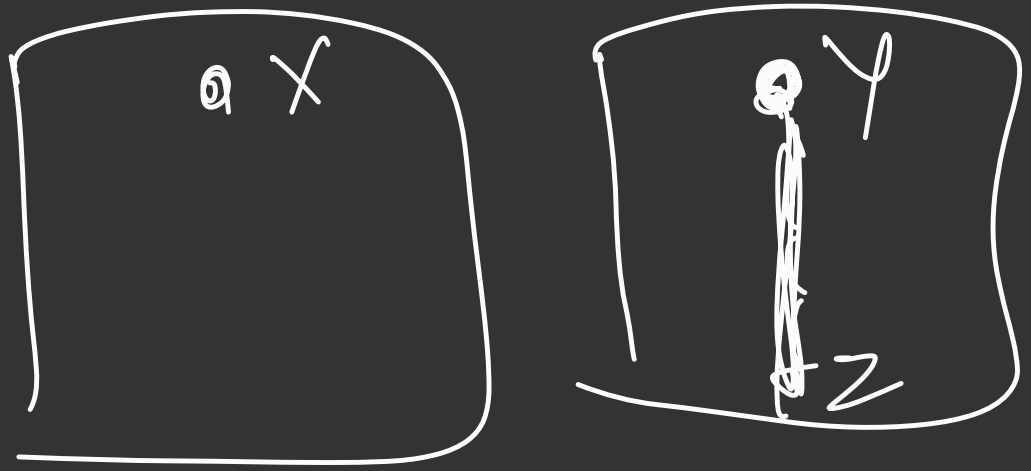
If you pick up any random node X and find out the farthest node from X, you are bound to end at one of the end points of the diameter

① choose a random node X

② find out farthest node Y from X

③ find out farthest node Z from Y

Y to Z is a diameter



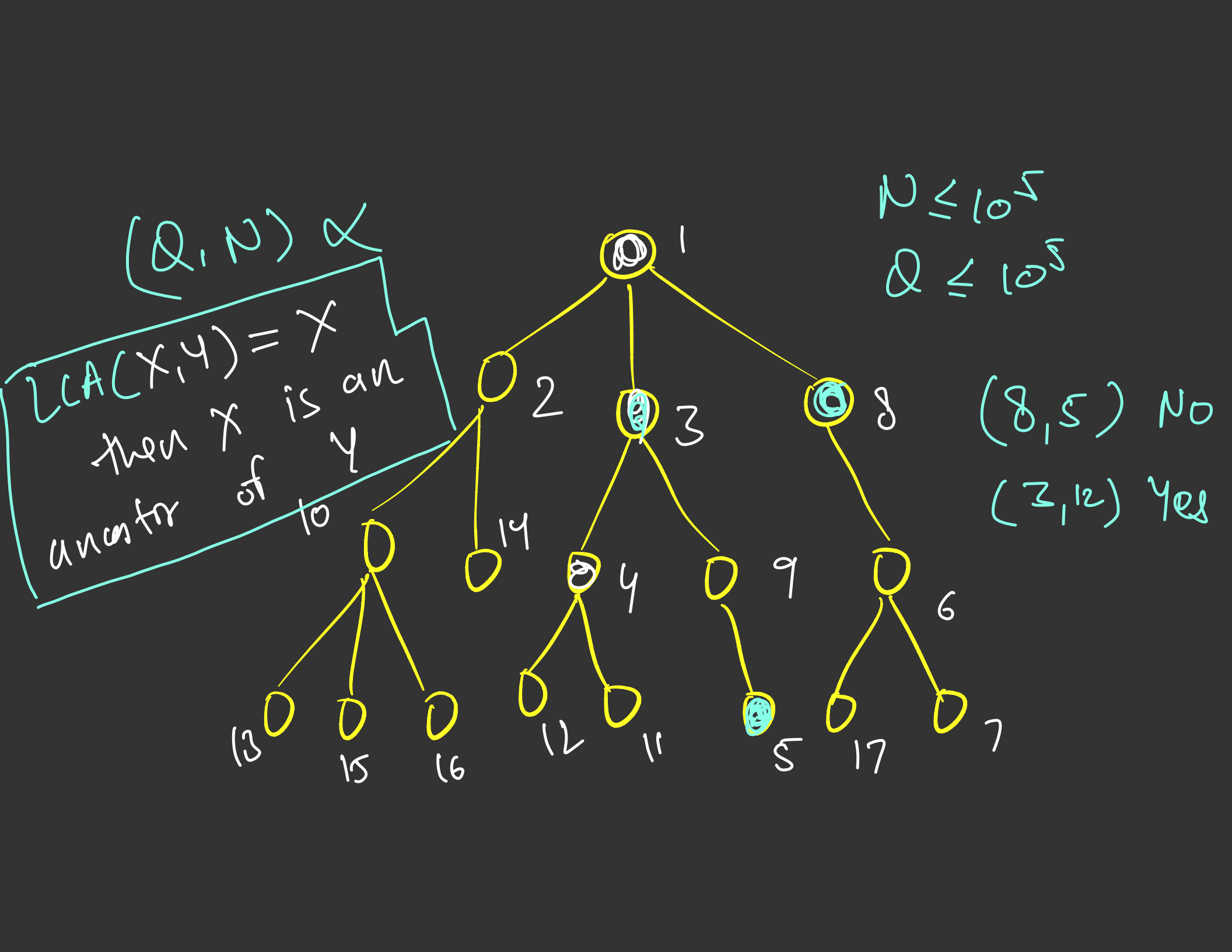
level(Z) = diameter

Ancestor - Descendant Problem

Given a rooted tree with N nodes and Q queries.

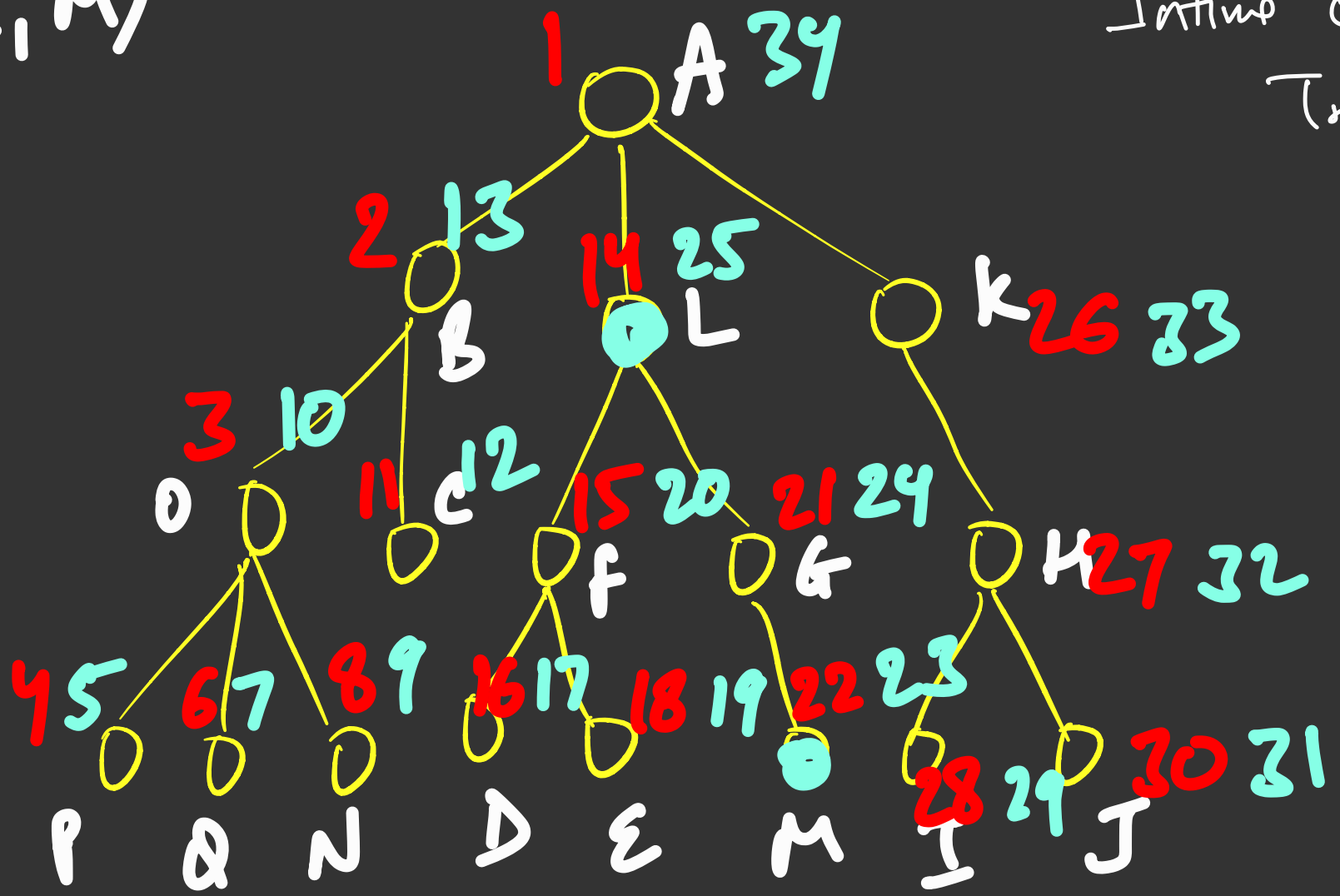
For each query of the form X, Y check whether X is an ancestor of Y or not

N <= 1e5, Q <= 1e5


(Q, N) Q
LCA(X,Y) = X
then X is an
ancestor of Y
N ≤ 10^5
Q ≤ 10^5
(8,5) No
(3,12) Yes
1
2
3
8
10
14
4
9
6
13
15
16
12
11
5
17
7

(L, M)

Intime outtime Trick



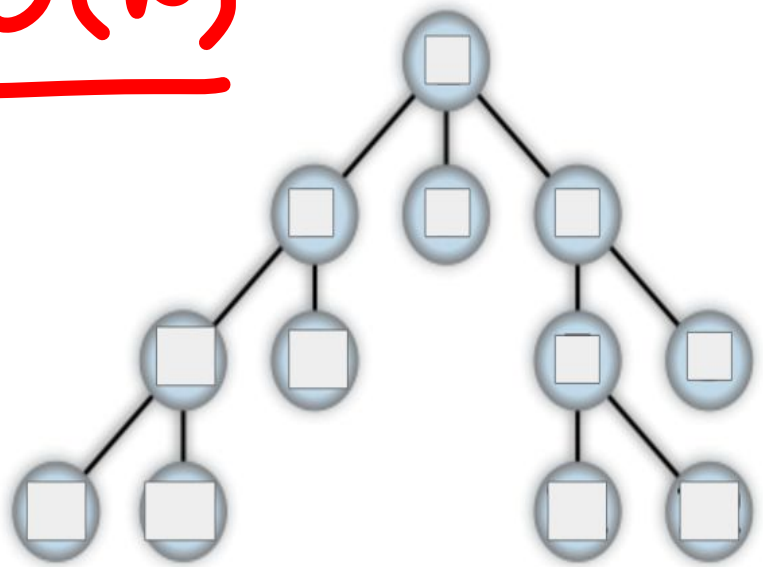
In - Out Time trick O(N)

Do a DFS traversal.

Store the following information for each node:

First visited time = In time

Last visited time = out time



Can you solve the ancestor descendant problem now?

In - Out Time trick

O(1)

Solving the ancestor - descendant problem:

If X is an ancestor of Y