Introduction to Trees 2

- Priyansh Agarwal

DFS Problem 1

N-1 edges
N nodes

You're given a tree with nodes numbered from 0 to N - 1, rooted at 0.

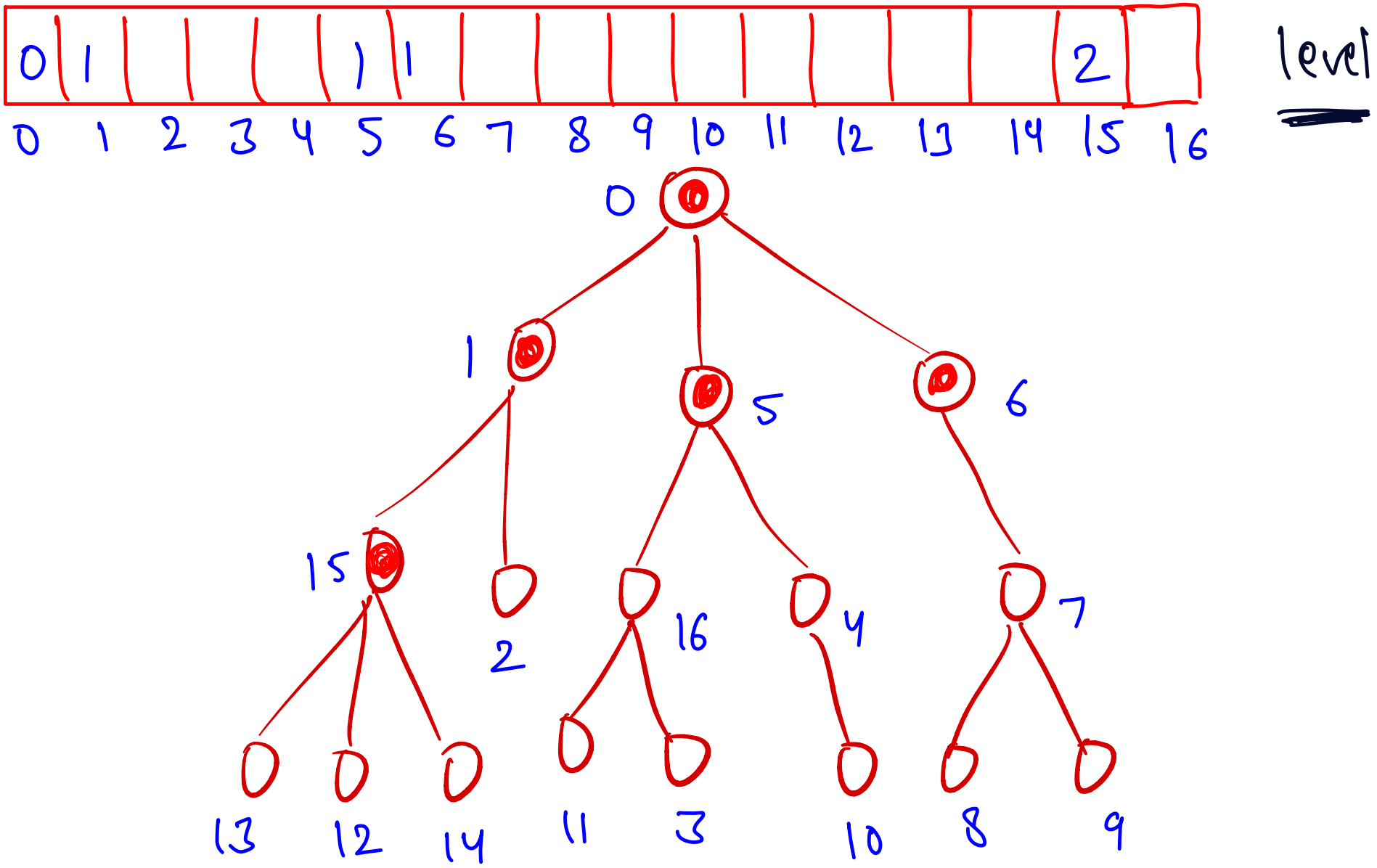
Find the level of each node assuming level of root = 0

level(x) = dist(root, x)

level(x) = k

level(p(x)) = k-1

level(c(x)) = k+1


0 1 1 2
level
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
0
1 5 6
15 2 16 4 7
13 12 14 11 3 10 8 9

DFS Problem 2

You're given a tree with nodes numbered from 0 to N - 1, rooted at 0.

Find the parent of each node assuming parent of root = -1

V. Easy Based on problem 1

DFS Problem 3

You're given a tree with nodes numbered from 0 to N - 1, rooted at 0.

Find the number of children of each node

① Find out no. of times we did the dfs call going down from the current node

②

edges[i].size() ⟶ no. of neighbours

- root node ⟶ edges[i].size(), non-root ⟶ edges[i].size() - 1

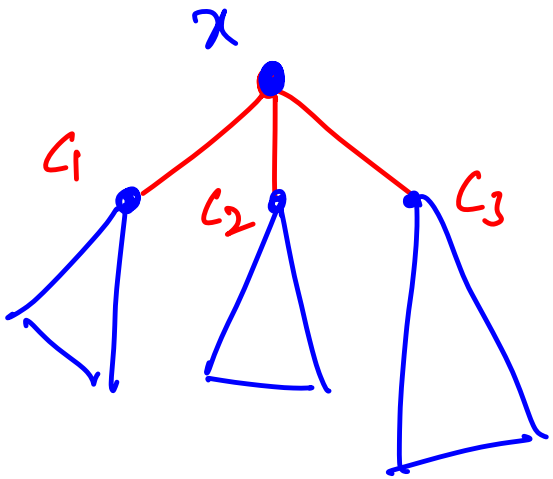
DFS Problem 4

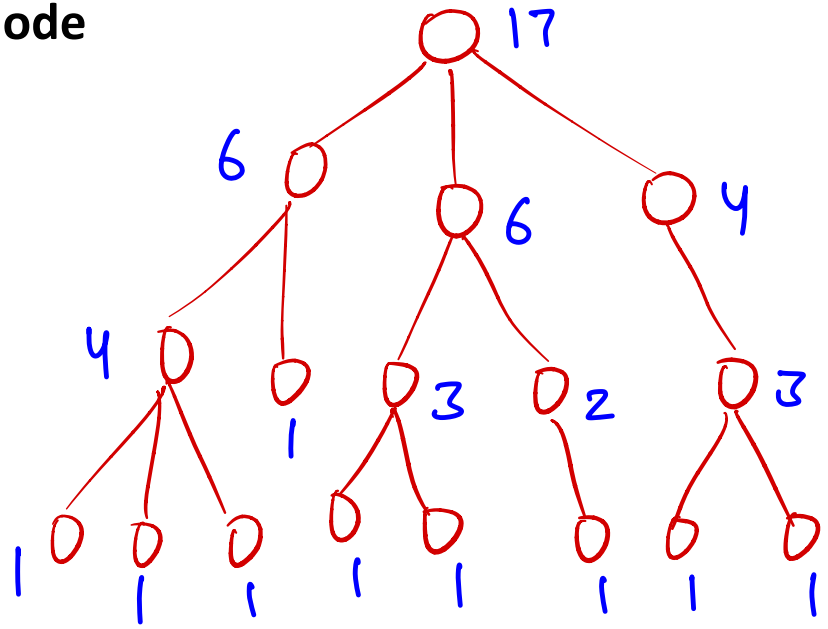
$sub[x] = 1 + \sum_{c_i \in children(x)} sub[c_i]$

$sub[leaf] = 1$

You're given a tree with nodes numbered from 0 to N - 1, rooted at 0.

Find the subtree size of each node





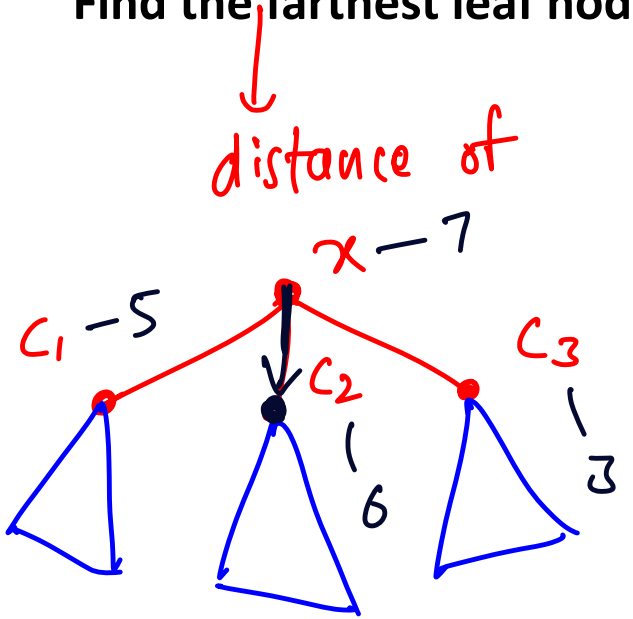
non leaf node ← $f(x) = 1 + \max(f(\text{child}))$

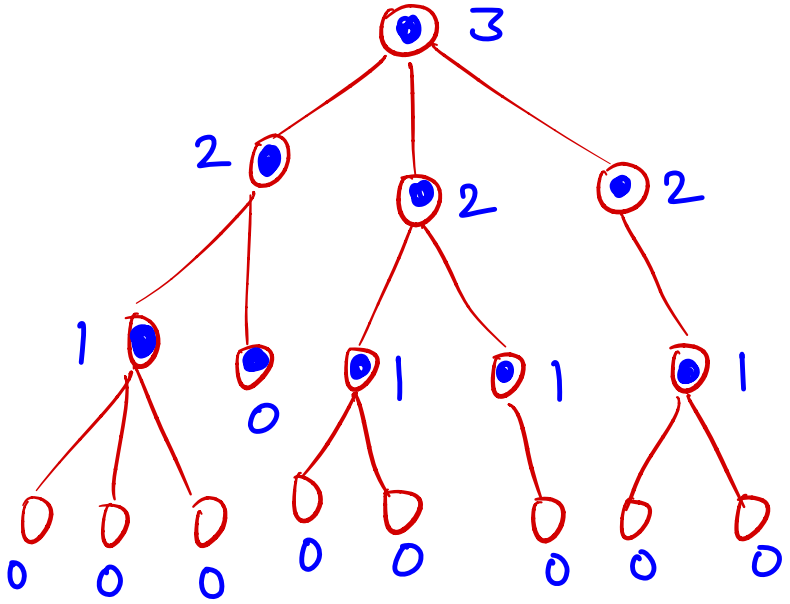
$f(\text{leaf}) = 0$

DFS Problem 5

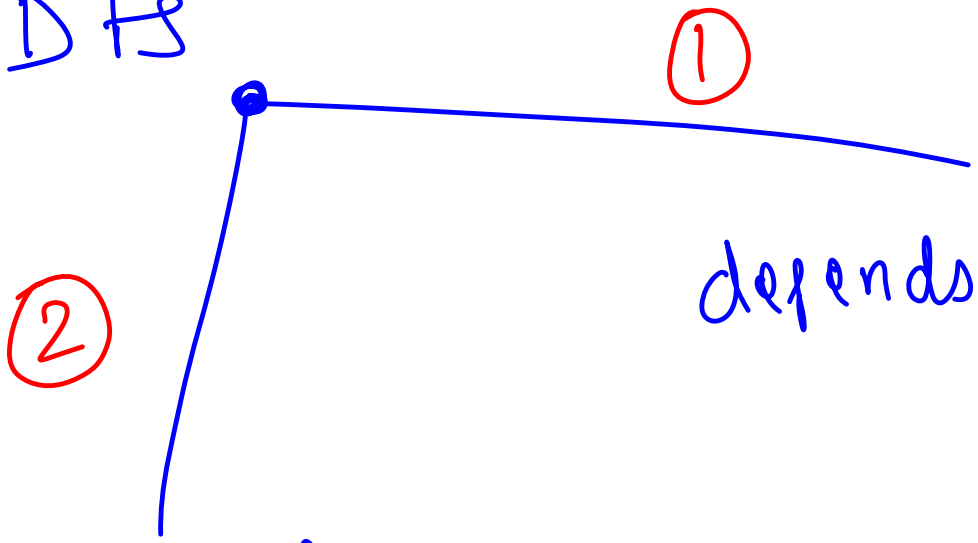
You're given a tree with nodes numbered from 0 to N - 1, rooted at 0.

Find the farthest leaf node from each node in its subtree





Dfs

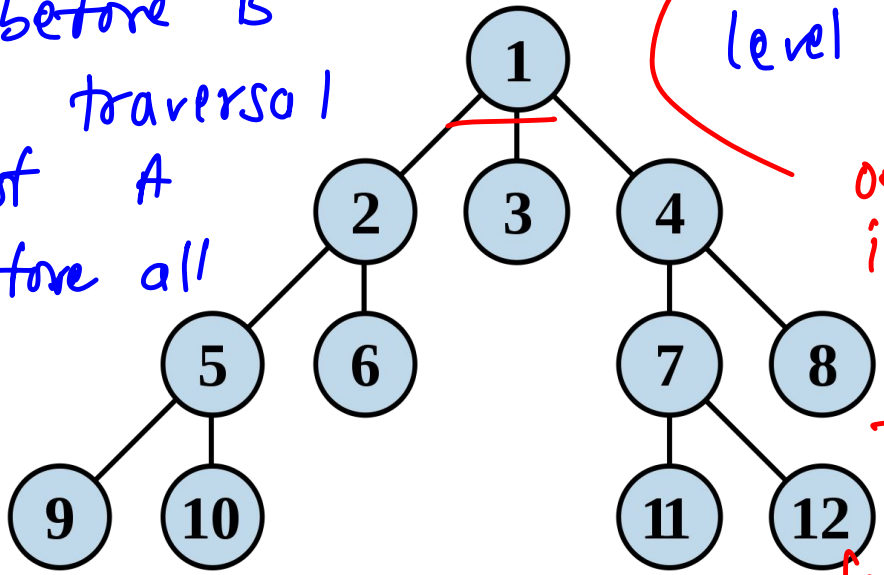


ans of curr node depends upon ans of ancestors

ans of curr node depends upon ans of descendants

BFS Traversal in a Tree

if A comes before B in the Bfs traversal all children of A must come before all children of B

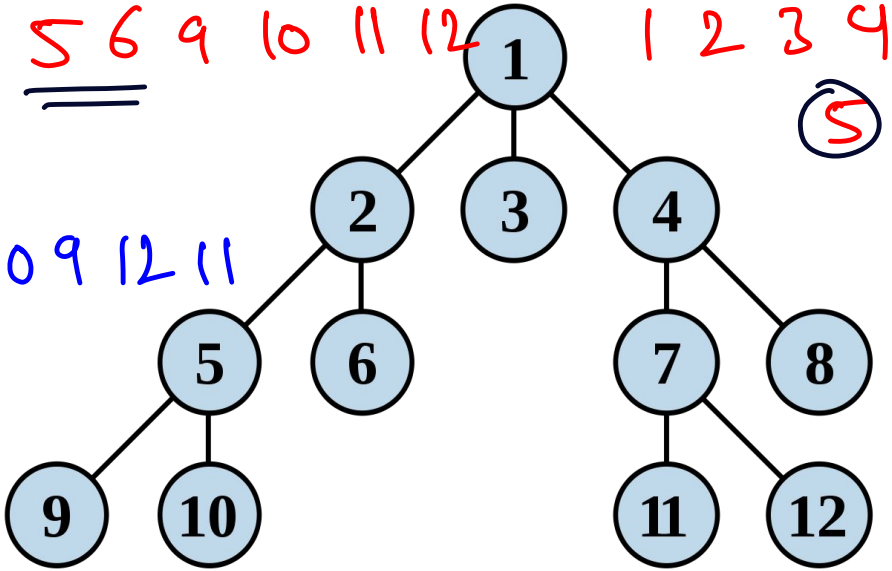


Bfs vs level order traversal

order of nodes in the same level matters for Bfs and does not matter for level order traversal

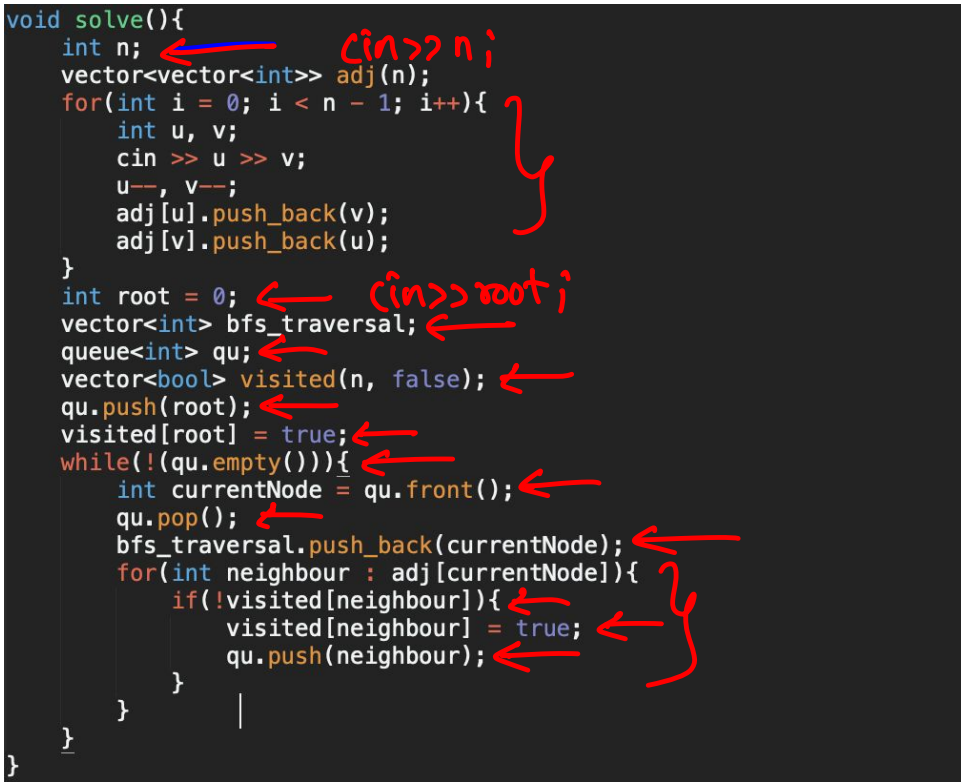
Nodes are numbered in the order in which they are visited

BFS Traversal in a Tree (Examples)



Nodes are numbered in the order in which they are visited

BFS Traversal in a Tree - Code O(N)




```
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);
            }
        }
    }
}
```

Diameter of a Tree (any 2 points which have the maximum distance b/w them)

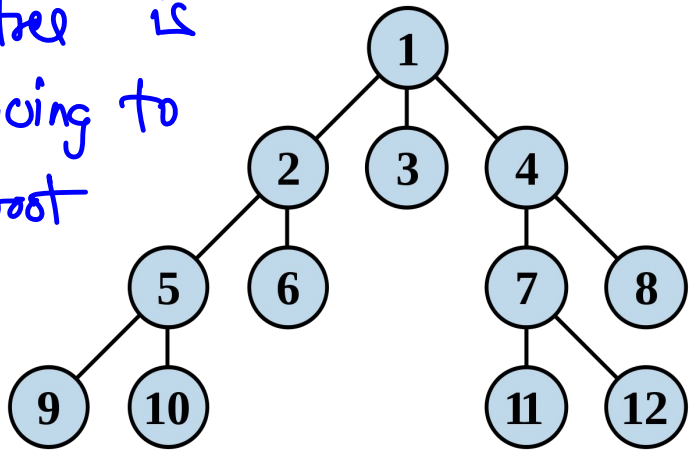
Diameter of a tree is the maximum distance between any 2 nodes in the tree.

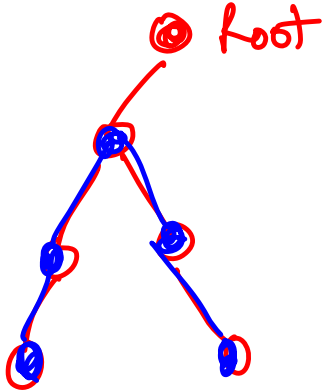
Example: Diameter of the tree below is 6 (9 to 12)

Note: 9 to 11, 10 to 11, 10 to 12 also form the diameters of the tree.

diameter of tree is not necessarily going to pass from the root

diameter is a path





Finding Diameter - Approach 1

- Find the longest path passing through each node in which that node is the LCA.
  - Longest path for X = Sum of distances of farthest 2 leaves in the subtree of X such that the 2 leaves are not chosen from the same child of X
- Diameter = length of longest path passing through any node in which that node is the LCA

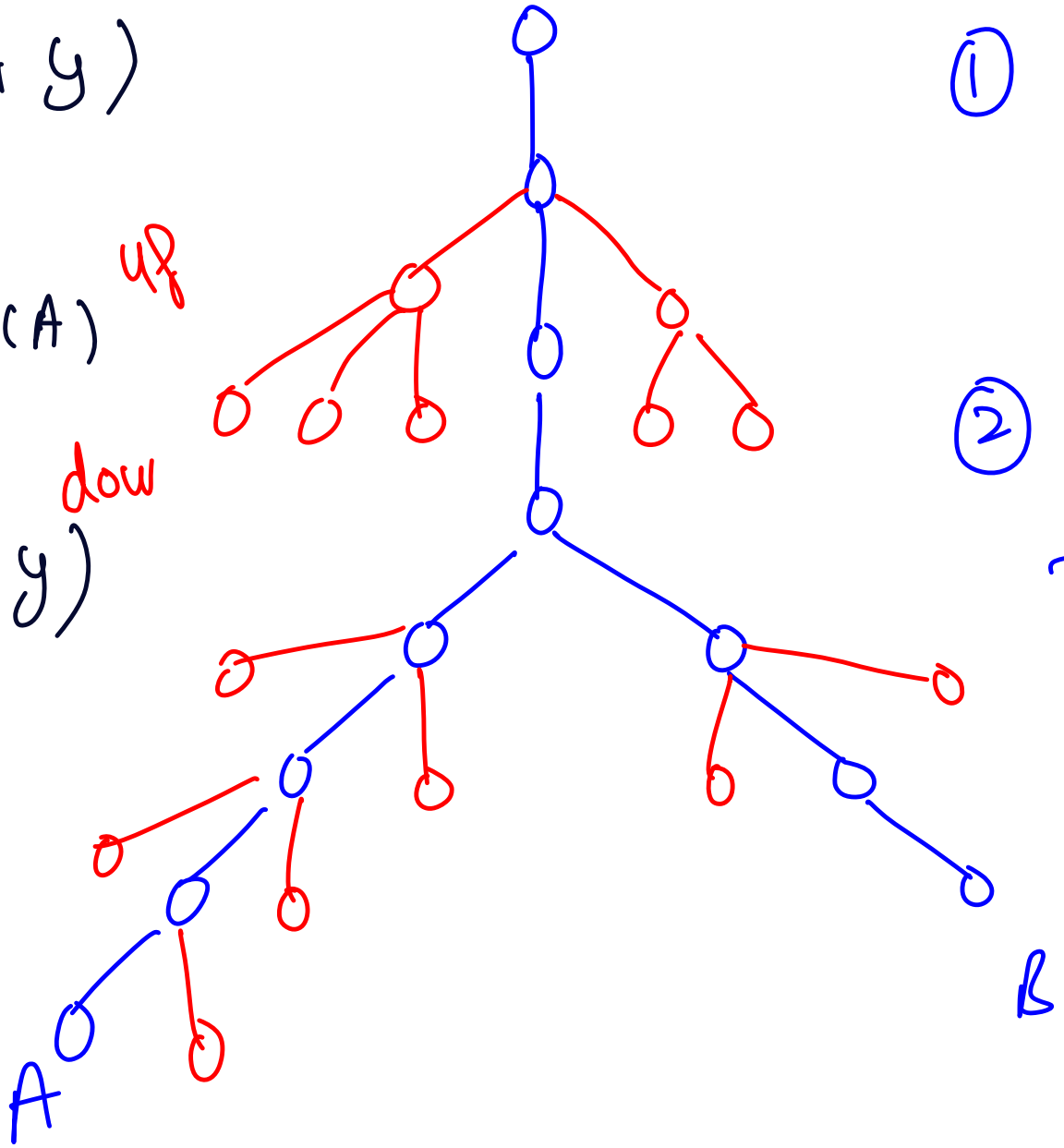
path (x, y)

↓

path (x, LCA) + path (LCA, y)

up

down



① path goes up and then comes down

② path goes up till LCA

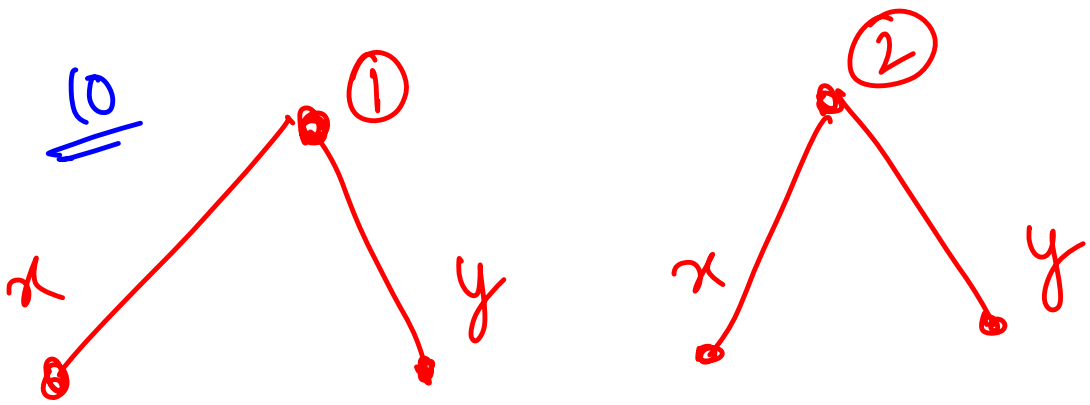
nodes → 1 2 3 4 ..... n

✓ ✓ ✓ ✓ ✓

LCA in some paths

LCA in some paths

m_3



$path_i \cap path_j = \phi$

$i \neq j$

$\bigcup_{j=1}^{N} path_j$ = all paths in the tree

path in which 1 is LCA and has maximum distance among all paths in which 1 is the LCA

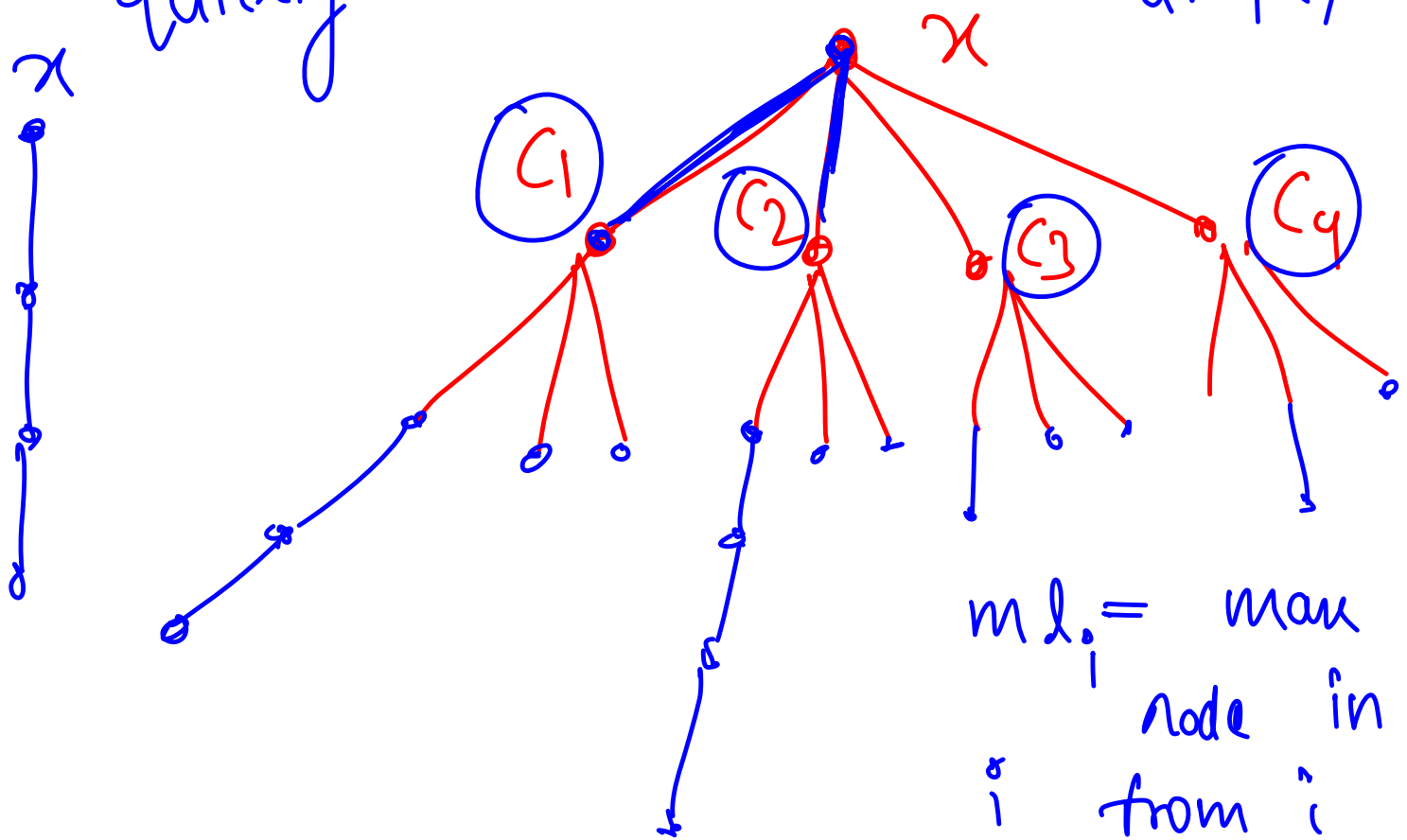
m_2

m_1

diameter = $\max(m_1, m_2 \cdots m_n)$

node ⟶ x

Find out the path with maximum distance such that x is the LCA in that path quickly



$ans[x] = md_{c_1} + md_{c_2} + 2$

$ans[x] = md_{c_1}$ (only one child) $+1$

md_i = max dist of a leaf node in the subtree of i from i

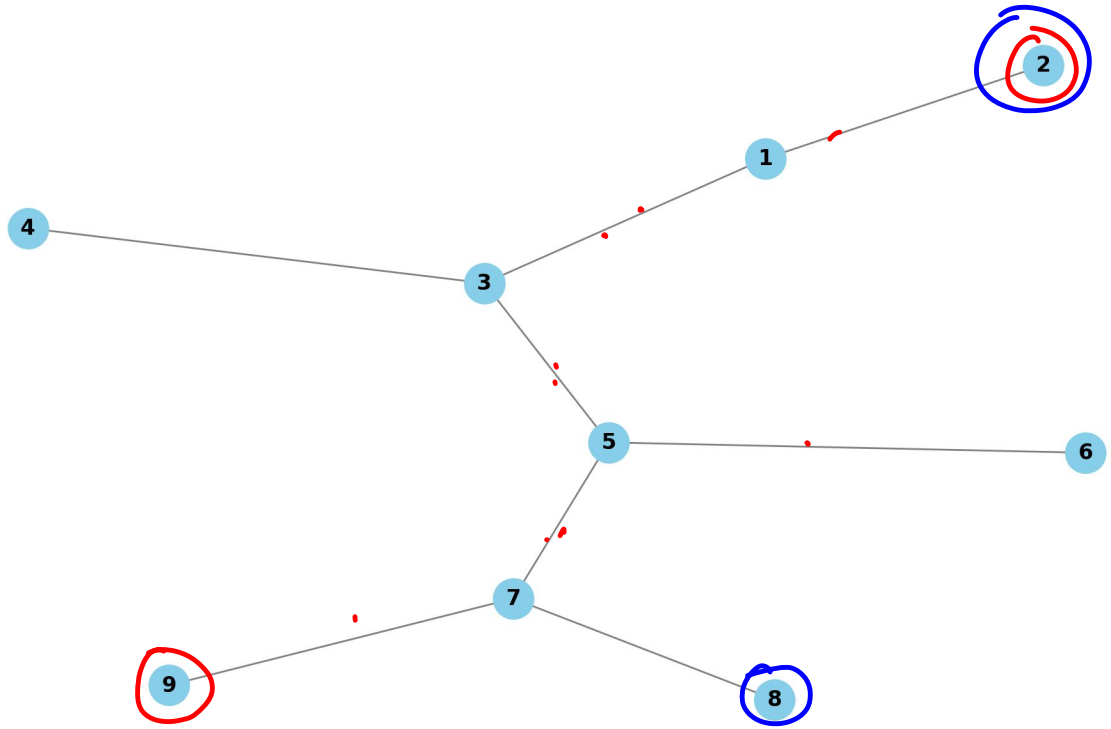
Rooted tree → farthest node from the deepest node in the tree

Finding Diameter - Approach 2

- Pick any node X
- Find the distance to every node in the tree from X and call it DistX[i]
  - DFS
- Find the node which has the highest value of DistX[i] and call it Y
  - If multiple such nodes exist, pick any of them
- Find the distance to every node in the tree from Y and call it DistY[i]
- Find the node which has the highest value of DistY[i] and call it Z
  - If multiple such nodes exist, pick any of them
- Diameter = Length of path between Y and Z

Visualising Approach 2

X	Y
4	9,8
3	9,8
5	2
7	2
8	2
9	2
6	2
1	9,8
2	9,8

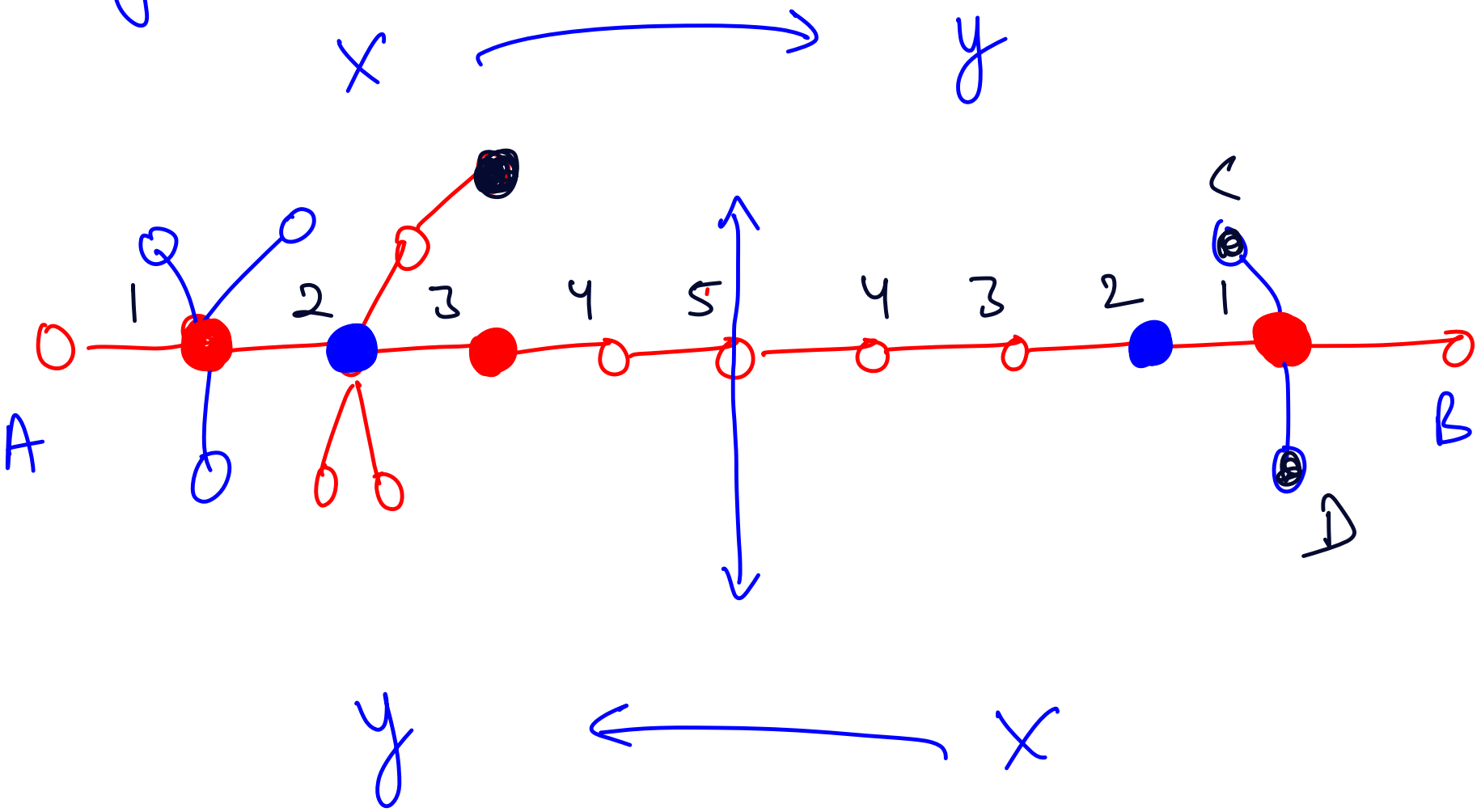


Idea of Proof of Correctness

- Node Y is bound to be one of the endpoints of a diameter.
- Node Z is bound to be the other endpoint of the diameter.

requires a proof

Assuming A, B is the diameter



Ancestor Problem

$x, y \longrightarrow LCA(x, y) = x$ → Can't solve it right now

You're given a tree with N nodes numbered from 0 to N - 1, rooted at 0.

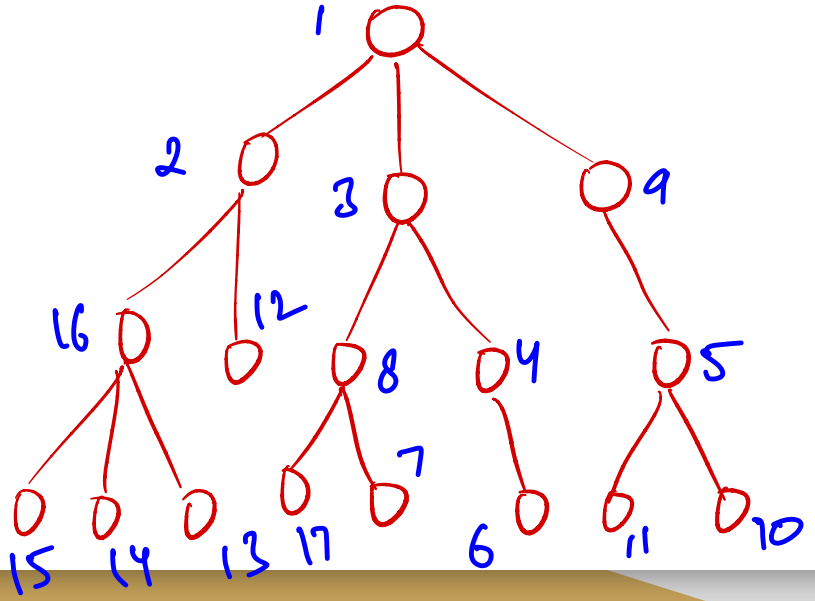
Process Q queries such that in each query you get X_i and Y_i. Find out if X_i is an ancestor of Y_i or not.

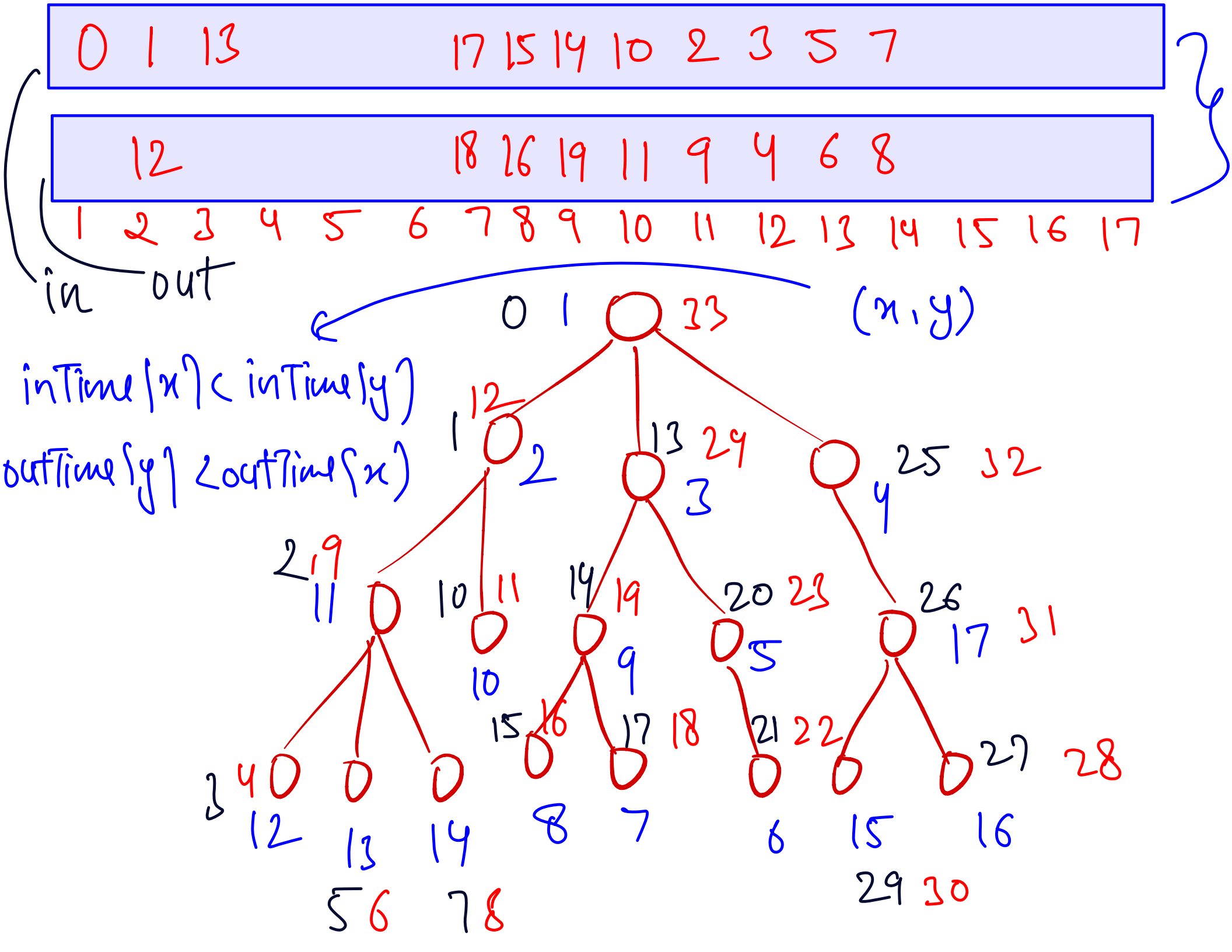
$1 <= N, Q <= 10^5$

(1, 12) → yes

(2, 17) → no

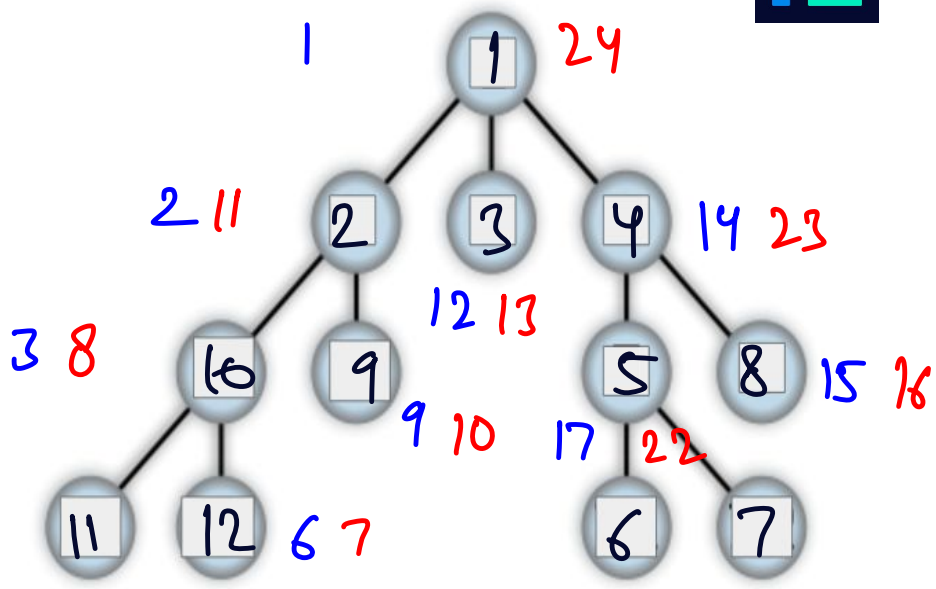
(8, 1) → no




0 1 13 17 15 14 10 2 3 5 7
12 18 16 19 11 9 4 6 8
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
in
out
(x, y)
inTime[x] < inTime[y]
outTime[y] < outTime[x]

In - Out Time trick

- Do a DFS traversal.
- Store the following information for each node:
- First visited time = In time
- Last visited time = out time



inTime[4] < inTime[6] < outTime[6] < outTime[4]

14 18 19 23

Solve the Ancestor Problem In - Out Time

If X is an ancestor of Y

$$X_{\text{in time}} < Y_{\text{in time}} < Y_{\text{out time}} < X_{\text{out time}}$$