Recitation 7

Trees

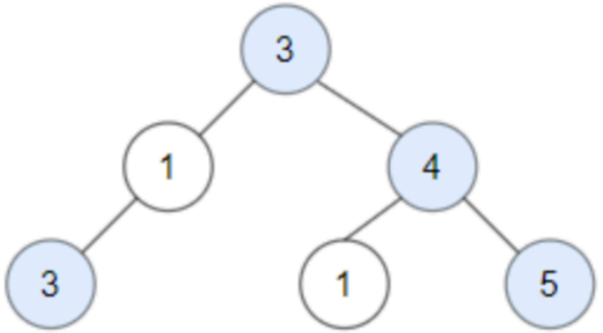
Adi Iyer

Q1

DFS

Given a binary tree root, a node X in the tree is named good if in the path from root to X there a re no nodes with a value greater than X. Return the number of good nodes in the binary tree.

Example 1:



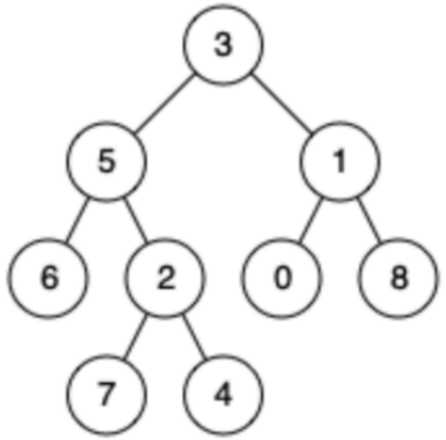

```
Input: root = [3,1,4,3,null,1,5]
Output: 4
Explanation: Nodes in blue are good.
Root Node (3) is always a good node.
Node 4 -> (3,4) is the maximum value in the path
starting from the root.
Node 5 -> (3,4,5) is the maximum value in the path
Node 3 -> (3,1,3) is the maximum value in the path.
```

```java
public class GoodNodesInBinaryTree {
    // DFS approach
    static int goodNodesDFS(TreeNode root, int maxSoFar) {
        if (root == null)
            return 0;

        int count = 0;
        if (root.val >= maxSoFar) {
            count++;
            maxSoFar = root.val;
        }

        count += goodNodesDFS(root.left, maxSoFar);
        count += goodNodesDFS(root.right, maxSoFar);

        return count;
    }
```

Q2

Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree. The lowest common ancestor is defined between two nodes p and q as the lowest node in T that has both p and q as descendants (where we allow a node to be a descendant of itself).

Example 1:




```
Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5,
q = 1
Output: 3
Explanation: The LCA of nodes 5 and 1 is 3.
```

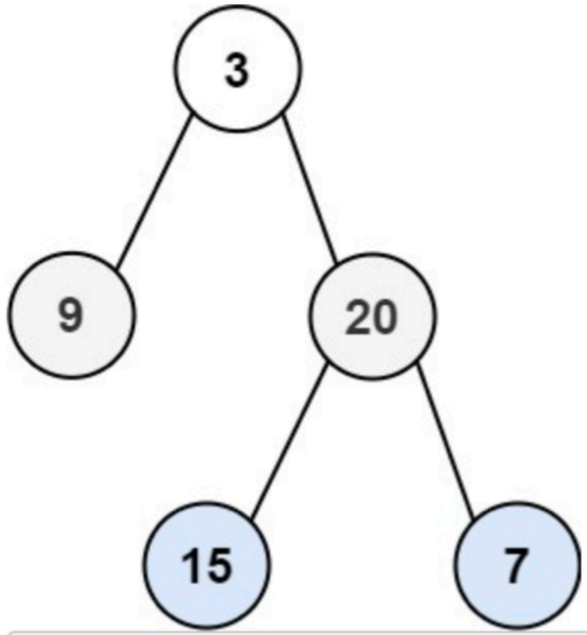
```java
    private boolean findPath(TreeNode root, List<Integer> path, int target) {
        if (root == null)
            return false;

        path.add(root.val);

        if (root.val == target)
            return true;

        if ((root.left != null && findPath(root.left, path, target)) ||
            (root.right != null && findPath(root.right, path, target)))
            return true;

        path.remove(path.size() - 1);
        return false;
    }

    // Main function to find the LCA
    public int findLCA(TreeNode root, int n1, int n2) {
        List<Integer> path1 = new ArrayList<>();
        List<Integer> path2 = new ArrayList<>();

        if (!findPath(root, path1, n1) || !findPath(root, path2, n2))
            return -1;

        int i;
        for (i = 0; i < path1.size() && i < path2.size(); i++) {
            if (!path1.get(i).equals(path2.get(i)))
                break;
        }

        return path1.get(i - 1);
    }
```

Q3

BFS

Given the root of a binary tree, return the zigzag level order traversal of its nodes' values. (i.e., fro m left to right, then right to left for the next level and alternate between).

Example 1:




```
Input: root = [3,9,20,null,null,15,7]
Output: [[3],[20,9],[15,7]]
```

```java
public class ZigzagLevelOrderTraversalBFS {
    public List<List<Integer>> zigzagLevelOrder(TreeNode root) {
        List<List<Integer>> result = new ArrayList<>();
        if (root == null) {
            return result;
        }

        Queue<TreeNode> queue = new LinkedList<>();
        queue.offer(root);
        boolean leftToRight = true;

        while (!queue.isEmpty()) {
            int size = queue.size();
            List<Integer> levelValues = new ArrayList<>();

            for (int i = 0; i < size; i++) {
                TreeNode node = queue.poll();
                if (leftToRight) {
                    levelValues.add(node.val);
                } else {
                    levelValues.add(0, node.val); // Add to the beginning for reverse order
                }

                if (node.left != null) {
                    queue.offer(node.left);
                }
                if (node.right != null) {
                    queue.offer(node.right);
                }
            }

            result.add(levelValues);
            leftToRight = !leftToRight;
        }

        return result;
    }
```

Q4

Problem 2 Report for every node v in tree T the length in edges of the longest path in v's subtree, storing the result for node v in $v.lp$.

This longest path could pass through node v or it might pass through a proper descendant x of v, as shown in the figure below.