https://course.acciojob.com/idle?question=a8342068-06f6-4331-963d-d9fca8776352

- EASY
- Max Score: 30 Points
-
-
-

Minimum Height of binary tree

You are given a pointer to the root of a binary tree. You have to find the minimum height of the binary tree.

NOTE A binary tree's minimum height is the number of nodes along the shortest path from the root node down to the closest leaf node.

You need to complete the given function. The input and printing of output will be handled by the driver code.

Input Format

The first line contains the number of test cases.

For each test case: You are given a pointer to the root of the binary tree.

Output Format

For each test case print the minimum height of the binary tree.

Example 1

Input:

```
1
```

Output

```
2
```

Explanation

```
     1
    /   \
  2      3
```

The minimum height is 2. Since there are two nodes in the shortest path.

Example 2

Input

```
1
```

Output

```
2
```

Explanation

```
     1
    /   \
  2      3
          /
 4
```

The minimum height is 2. Since there are two nodes in the shortest path 1->2.

Constraints

1 <= T <= 10

1 <= N <= 10000

Topic Tags

- **Recursion**
- **Trees**

My code

```
// in java
import java.util.LinkedList;
import java.util.Queue;
import java.io.*;
import java.util.*;
import java.lang.*;

class Node{
    int data;
    Node left;
    Node right;
    Node(int data){
        this.data = data;
        left=null;
        right=null;
    }
}

class Main {

    static Node buildTree(String str){

        if(str.length()==0 || str.charAt(0)=='N'){
            return null;
        }
        String ip[] = str.split(" ");
        Node root = new Node(Integer.parseInt(ip[0]));
        Queue<Node> queue = new LinkedList<>();
        queue.add(root);
        int i = 1;
        while(queue.size()>0 && i < ip.length) {
            Node currNode = queue.peek();
            queue.remove();
            String currVal = ip[i];
            if(!currVal.equals("N")) {
                currNode.left = new Node(Integer.parseInt(currVal));
                queue.add(currNode.left);
            }
            i++;
            if(i >= ip.length)
                break;
            currVal = ip[i];
            if(!currVal.equals("N")) {
                currNode.right = new Node(Integer.parseInt(currVal));
                queue.add(currNode.right);
            }
            i++;
        }
        return root;
    }
    static void printInorder(Node root){
        if(root == null)
            return;

        printInorder(root.left);
        System.out.print(root.data+" ");
        printInorder(root.right);
    }

        public static void main (String[] args) throws IOException{
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        int t=Integer.parseInt(br.readLine());
        while(t > 0){
            String s = br.readLine();
                Node root = buildTree(s);
            Solution ob = new Solution();
                System.out.println(ob.height(root));
            t--;
        }
    }
}

class Solution {
    int height(Node node)
    {
            Node root=node;
        if(root==null) return 0;
              if(root.left==null&&root.right==null)
                      return 1;

          if(root.left!=null&&root.right!=null)
                  return 1+Math.min(height(root.left),height(root.right));


        if(root.right!=null)
                return 1+height(root.right);
        return 1+height(root.left);

    }
}
/*int minDepth(Node root)
    {
          if(root==null) return 0;
        if(root.left==null&&root.right==null) return 1;
        if(root.left!=null&&root.right!=null) return
1+Math.min(minDepth(root.left),minDepth(root.right));
        if(root.right!=null) return 1+minDepth(root.right);
        return 1+minDepth(root.left);
    }*/
```