https://course.acciojob.com/idle?question=d825cc36-0ad2-49c8-92d8-c290fef697ee

MEDIUM

Max Score: 40 Points

Subtree or not

You are given a pointer to the head of two binary trees. You have to return if the second tree is a subtree of the first or not.

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 head of two binary trees.

Output Format

For each test case return true if the second tree is a subtree of the first tree otherwise false.

Example 1

Input

```
1
3

      1
    /   \
   2     3

1
     3
```

Output

```
true
```

Explanation

Since '3' is a subtree of the first tree therefore ans is true.

Example 2

Input

```
1
4

      1
    /   \
   2     3
 /
4

2
     2
    /
   4
```

Output

```
true
```

Explanation

'2', '4' is a subtree of first tree.

Constraints

1 <= T <= 10

1 <= N <= 1000

1 <= M <= 1000

Topic Tags

Recursion

Trees

My code

```
import java.util.LinkedList;
import java.util.Queue;
import java.io.*;
import java.util.*;

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 tt= br.readLine();
                Node rootT = buildTree(tt);

                String s= br.readLine();
                Node rootS = buildTree(s);
                Solution tr=new Solution();
                boolean st=tr.isSubtree(rootT, rootS);
                if(st==true){
                    System.out.println("true");
                }else{
                    System.out.println("false");
                }
            }
        }
}
class Solution {
    public boolean isSubtree(Node s, Node t) {
        boolean b = false;
        if(s ==null && t == null){
            return true;
        }

        Queue<Node> q = new LinkedList<>();
        q.offer(s);
        while(!q.isEmpty()){
            Node temp =q.poll();
            if(temp.data == t.data){
                if(same(temp,t)){
                    b = true;
```

```
                break;
            }
        }
        if(temp.left!=null)
        q.offer(temp.left);
        if(temp.right!=null){
            q.offer(temp.right);
        }
    }
    return b;
  }
  boolean same(Node root1,Node root2){
    if(root1 ==null && root2 == null){
        return true;
    }
    if(root1 ==null || root2 == null){
        return false;
    }
    return root1.data == root2.data && same(root1.left,root2.left)
&& same(root1.right,root2.right);
  }
}
```