1. Structurally Identical (Binary Tree)

Given two trees check if they are structurally identically. Structurally identical trees are trees that have structure. They may or may not have the same data though.

Input Format

Enter the values of all the nodes in the binary tree in pre-order format where true suggest the node ex false suggests it is NULL

Constraints

```
1 <= N <= 10^4
```

Output Format

Display the Boolean result

Sample Input

```
10 true 20 true 40 false false true 50 false false
10 true 20 true 40 false false true 50 false false
```

Sample Output

```
true
```

Explanation

The given two trees have the exact same structure and hence we print true.

```java
import java.util.*;
public class Main {

    static Scanner scn = new Scanner(System.in);

    public static void main(String[] args) {
        Main m = new Main();
        BinaryTree bt1 = m.new BinaryTree();
        BinaryTree bt2 = m.new BinaryTree();
        System.out.println(bt1.structurallyIdentical(bt2));
    }
```

```
    private class BinaryTree {
        private class Node {
            int data;
            Node left;
            Node right;
        }

        private Node root;
        private int size;

        public BinaryTree() {
            this.root = this.takeInput(null, false);
        }

        public Node takeInput(Node parent, boolean ilc) {

            int cdata = scn.nextInt();
            Node child = new Node();
            child.data = cdata;
            this.size++;

            // left
            boolean hlc = scn.nextBoolean();

            if (hlc) {
                child.left = this.takeInput(child, true);
            }

            // right
            boolean hrc = scn.nextBoolean();

            if (hrc) {
                child.right = this.takeInput(child, false);
            }

            // return
            return child;
        }

        public boolean structurallyIdentical(BinaryTree other) {
            return this.structurallyIdentical(this.root, other.root);
        }

        private boolean structurallyIdentical(Node tnode, Node onode) {
            // write your code here
            if(tnode != null && onode == null || tnode == null && onode !=
null){
                return false;
```

```
            }
            if(tnode == null && onode == null){
                return true;
            }
            boolean left = structurallyIdentical(tnode.left,onode.left);
            boolean right = structurallyIdentical(tnode.right,onode.right);
            return left && right;
        }

    }
}
```

2. Level Order (Zigzag , Binary Tree)

Given a binary tree. Print the zig zag order i.e print level 1 from left to right, level 2 from right
means odd levels should get printed from left to right and even levels should be printed fro

Input Format

Enter the values of all the nodes in the binary tree in pre-order format where true suggest t
false suggests it is NULL

Constraints

None

Output Format

Display the values in zigzag level order in which each value is separated by a space

Sample Input

```
10 true 20 true 40 false false true 50 false false
```

Sample Output

```
10 30 20 40 50 60 73
```

```java
import java.util.*;
public class Main {

    static Scanner scn = new Scanner(System.in);

    public static void main(String[] args) {
        Main m = new Main();
        BinaryTree bt = m.new BinaryTree();
        bt.levelOrderZZ();
    }

    private class BinaryTree {
        private class Node {
            int data;
            Node left;
            Node right;
        }

        private Node root;
        private int size;

        public BinaryTree() {
            this.root = this.takeInput(null, false);
        }

        public Node takeInput(Node parent, boolean ilc) {

            int cdata = scn.nextInt();
            Node child = new Node();
            child.data = cdata;
            this.size++;

            // left
            boolean hlc = scn.nextBoolean();

            if (hlc) {
                child.left = this.takeInput(child, true);
            }

            // right
            boolean hrc = scn.nextBoolean();

            if (hrc) {
                child.right = this.takeInput(child, false);
            }

            // return
            return child;
```

```
        }

        public void levelOrderZZ() {
            Queue<Node>q = new LinkedList<>();
            q.add(this.root);
            boolean even = false;
            while(!q.isEmpty()){
                int size = q.size();
                Node []arr = new Node[size];
                for(int i=0;i<size;i++){
                    Node rv = q.remove();
                    if(rv.left != null){
                        q.add(rv.left);
                    }
                    if(rv.right != null){
                        q.add(rv.right);
                    }
                    arr[i] = rv;
                }
                if(even){
                    for(int i=arr.length-1; i>=0;i--){
                        System.out.print(arr[i].data+" ");
                    }
                    even = false;
                }
                else{
                    for(int i=0; i<arr.length;i++){
                        System.out.print(arr[i].data+" ");
                    }
                    even = true;
                }
            }
            // write your code here
        }

    }

}
```

3. **ArrayList of levels (Binary Tree)**

Given a Binary tree, write code to create a separate array list for each level. You sh
of arraylist.

Input Format

Enter the values of all the nodes in the binary tree in pre-order format where true su
and false suggests it is NULL

Constraints

None

Output Format

Display the resulting arraylist of arraylist according to given sample examples.

Sample Input

```
50 true 12 true 18 false false true 13 false fa
```

Sample Output

```
[[50], [12], [18, 13]]
```

```java
import java.util.*;
public class Main {

    static Scanner scn = new Scanner(System.in);

    public static void main(String[] args) {
        Main m = new Main();
        BinaryTree bt1 = m.new BinaryTree();
        System.out.println(bt1.levelArrayList());
    }

    private class BinaryTree {
        private class Node {
            int data;
            Node left;
            Node right;
        }
```

```
        private Node root;
        private int size;

        public BinaryTree() {
            this.root = this.takeInput(null, false);
        }

        public Node takeInput(Node parent, boolean ilc) {

            int cdata = scn.nextInt();
            Node child = new Node();
            child.data = cdata;
            this.size++;

            // left
            boolean hlc = scn.nextBoolean();

            if (hlc) {
                child.left = this.takeInput(child, true);
            }

            // right
            boolean hrc = scn.nextBoolean();

            if (hrc) {
                child.right = this.takeInput(child, false);
            }

            // return
            return child;
        }

        public ArrayList<ArrayList<Integer>> levelArrayList() {
            Queue<Node>q = new LinkedList<>();
            q.add(this.root);
            ArrayList<ArrayList<Integer>> levelorder = new
ArrayList<ArrayList<Integer>>();
            while(!q.isEmpty()){
                int size = q.size();
                ArrayList<Integer>li = new ArrayList<>();
                for(int i=0;i<size;i++){
                    Node rv = q.remove();
                    if(rv.left != null){
                        q.add(rv.left);
                    }
                    if(rv.right != null){
                        q.add(rv.right);
```

```
                    }
                    li.add(rv.data);
                }
                    levelorder.add(li);
            }
            return levelorder;
            // write your code here
        }

    }

}
```

4. Is Balanced (Binary Tree)

Given a Binary tree check if it is balanced i.e. depth of the left and right sub-trees of every no

Input Format

Enter the values of all the nodes in the binary tree in pre-order format where true suggest the
false suggests it is NULL

Constraints

1 <= No of nodes <= 10^5

Output Format

Display the Boolean result

Sample Input

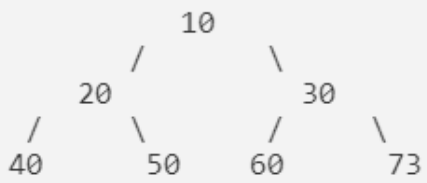
```
10 true 20 true 40 false false true 50 false false
```

Sample Output

```
true
```

Explanation

The tree looks like



The given tree is clearly balanced as depths of the left and right sub-trees of every node differ by 1 or less.

```java
import java.util.*;
public class Main {

    static Scanner scn = new Scanner(System.in);

    public static void main(String[] args) {
        Main m = new Main();
        BinaryTree bt = m.new BinaryTree();
```

```
        System.out.println(bt.isBalanced());
    }

    private class BinaryTree {
        private class Node {
            int data;
            Node left;
            Node right;
        }

        private Node root;
        private int size;

        public BinaryTree() {
            this.root = this.takeInput(null, false);
        }

        public Node takeInput(Node parent, boolean ilc) {

            int cdata = scn.nextInt();
            Node child = new Node();
            child.data = cdata;
            this.size++;

            // left
            boolean hlc = scn.nextBoolean();

            if (hlc) {
                child.left = this.takeInput(child, true);
            }

            // right
            boolean hrc = scn.nextBoolean();

            if (hrc) {
                child.right = this.takeInput(child, false);
            }

            // return
            return child;
        }

        public boolean isBalanced() {
            return this.isBalanced(this.root).isBalanced;
        }

        private BalancedPair isBalanced(Node node) {
            // write your code here
```

```
            if(node == null)
                return new BalancedPair();
            BalancedPair lbp = isBalanced(node.left);
            BalancedPair rbp = isBalanced(node.right);
            BalancedPair nbp = new BalancedPair();
            nbp.height = Math.max(lbp.height,rbp.height)+1;
            if(Math.max(lbp.height,rbp.height) -
Math.min(lbp.height,rbp.height) == 0)
                nbp.isBalanced = true;
            else
                nbp.isBalanced = false;
            return nbp;
        }

        private class BalancedPair {
            int height;
            boolean isBalanced;
        }

    }
}
```

5. **Level Order (New Line, Binary Tree)**

Given a binary tree. Print the level order traversal, make sure each level start at a new line.

Input Format

Enter the values of all the nodes in the binary tree in pre-order format where true suggest the
false suggests it is NULL

Constraints

None

Output Format

Print the level order in which the different levels are in different lines, all the values should be in
manner

Sample Input

```
10 true 20 true 40 false false true 50 false false
```

Sample Output

```
10
20 30
40 50 60 73
```

```java
import java.util.*;
public class Main {

    static Scanner scn = new Scanner(System.in);

    public static void main(String[] args) {
        Main m = new Main();
        BinaryTree bt = m.new BinaryTree();
        bt.levelOrderNewLine();
    }

    private class BinaryTree {
        private class Node {
            int data;
            Node left;
            Node right;
        }
```

```
    private Node root;
    private int size;

    public BinaryTree() {
        this.root = this.takeInput(null, false);
    }

    public Node takeInput(Node parent, boolean ilc) {

        int cdata = scn.nextInt();
        Node child = new Node();
        child.data = cdata;
        this.size++;

        // left
        boolean hlc = scn.nextBoolean();

        if (hlc) {
            child.left = this.takeInput(child, true);
        }

        // right
        boolean hrc = scn.nextBoolean();

        if (hrc) {
            child.right = this.takeInput(child, false);
        }

        // return
        return child;
    }

    public void levelOrderNewLine() {
        Queue<Node>q = new LinkedList<>();
        q.add(this.root);
        while(!q.isEmpty()){
            int size = q.size();
            for(int i=0;i<size;i++){
                Node rv = q.remove();
                System.out.print(rv.data+" ");
                if(rv.left != null){
                    q.add(rv.left);
                }
                if(rv.right != null){
                    q.add(rv.right);
                }
            }
```

```
                System.out.println();
            }
            // write your code here
        }

    }

}
```

6. Remove the leaves (Binary Tree)

Given a binary tree, remove all the leaves from the tree

Input Format

Enter the value of the nodes of the tree

Constraints

None

Output Format

Display the tree in which all the leaves have been removed in pre-order traversa
Left->data => Root->data <= Right->Data
Output END if left or right node is NULL

Sample Input

```
50 true 12 true 18 false false false false
```

Sample Output

```
12 => 50 <= END
END => 12 <= END
```

```java
import java.util.*;
public class Main {

    static Scanner scn = new Scanner(System.in);

    public static void main(String[] args) {
        Main m = new Main();
        BinaryTree bt = m.new BinaryTree();
        bt.removeLeaves();
        bt.display();
    }

    private class BinaryTree {
        private class Node {
            int data;
            Node left;
            Node right;
        }

        private Node root;
        private int size;

        public BinaryTree() {
            this.root = this.takeInput(null, false);
        }

        public Node takeInput(Node parent, boolean ilc) {

            int cdata = scn.nextInt();
            Node child = new Node();
            child.data = cdata;
            this.size++;

            // left
            boolean hlc = scn.nextBoolean();

            if (hlc) {
                child.left = this.takeInput(child, true);
            }

            // right
            boolean hrc = scn.nextBoolean();

            if (hrc) {
                child.right = this.takeInput(child, false);
            }
```

```
            // return
            return child;
        }

        public void display() {
            this.display(this.root);
        }

        private void display(Node node) {
            if (node == null) {
                return;
            }

            String str = "";

            if (node.left != null) {
                str += node.left.data;
            } else {
                str += "END";
            }

            str += " => " + node.data + " <= ";

            if (node.right != null) {
                str += node.right.data;
            } else {
                str += "END";
            }

            System.out.println(str);

            this.display(node.left);
            this.display(node.right);
        }

        public void removeLeaves() {
            this.removeLeaves(null, this.root, false);
        }

        private void removeLeaves(Node parent, Node child, boolean ilc) {

            if(child == null)
                return;
            if(child.left != null && child.left.left == null &&
child.left.right == null){
                child.left = null;
            }
```

```
            if(child.right != null && child.right.left == null &&
child.right.right == null){
                child.right = null;
            }
            removeLeaves(parent,child.left,ilc);
            removeLeaves(parent,child.right,ilc);

            // write your code here
        }

    }

}
```

7. Sum of the nodes (Binary tree)

Given a binary tree find sum of all the nodes.

Input Format

Enter the value of the node then the Boolean choice i.e whether the node has a le
child's data . The input acts in a recursive manner i.e when the left child's children
onto the parent's right child

Constraints

None

Output Format

Display the sum of all the nodes

Sample Input

```
50 true 25 true 12 false false false true 75 true 6
```

Sample Output

```
224
```

Explanation

If we enter the following values

50 true 25 true 12 false false false true 75 true 62 false false false

the tree would look like :

25 => 50 <= 75

12 => 25 <= END

END => 12 <= END

62 => 75 <= END

END => 62 <= END

```java
import java.util.*;
public class Main {

    static Scanner scn = new Scanner(System.in);

    public static void main(String[] args) {
        Main m = new Main();
        BinaryTree bt = m.new BinaryTree();
        System.out.println(bt.sumOfNodes());
    }

    private class BinaryTree {
        private class Node {
            int data;
            Node left;
            Node right;
        }

        private Node root;
        private int size;

        public BinaryTree() {
            this.root = this.takeInput(null, false);
        }

        public Node takeInput(Node parent, boolean ilc) {

            int cdata = scn.nextInt();
            Node child = new Node();
            child.data = cdata;
            this.size++;

            // left
            boolean hlc = scn.nextBoolean();

            if (hlc) {
                child.left = this.takeInput(child, true);
            }

            // right
            boolean hrc = scn.nextBoolean();

            if (hrc) {
                child.right = this.takeInput(child, false);
            }

            // return
            return child;
```

```
        }

        public int sumOfNodes() {
            return this.sumOfNodes(this.root) ;
        }

        private int sumOfNodes(Node node) {
            if(node == null){
                return 0;
            }
            int sum = node.data + sumOfNodes(node.left) +
sumOfNodes(node.right);
            // write your code here
            return sum;
        }
    }
}
```

8. Sibling (Binary Tree)

Given a binary tree print all nodes that don't have a sibling

Input Format

Enter the values of all the nodes in the binary tree in pre-order format where true suggest the no
false suggests it is NULL

Constraints

None

Output Format

Display all the nodes which do not have a sibling in a space separated manner

Sample Input

```
50 true 12 true 18 false false false false
```

Sample Output

```
12 18
```

```java
import java.util.*;
public class Main {

    static Scanner scn = new Scanner(System.in);

    public static void main(String[] args) {
        Main m = new Main();
        BinaryTree bt = m.new BinaryTree();
        bt.sibling();
    }

    private class BinaryTree {
        private class Node {
            int data;
            Node left;
            Node right;
        }

        private Node root;
        private int size;
```

```java
    public BinaryTree() {
        this.root = this.takeInput(null, false);
    }

    public Node takeInput(Node parent, boolean ilc) {

        int cdata = scn.nextInt();
        Node child = new Node();
        child.data = cdata;
        this.size++;

        // left
        boolean hlc = scn.nextBoolean();

        if (hlc) {
            child.left = this.takeInput(child, true);
        }

        // right
        boolean hrc = scn.nextBoolean();

        if (hrc) {
            child.right = this.takeInput(child, false);
        }

        // return
        return child;
    }

    public void sibling() {
        this.sibling(this.root);
    }

    private void sibling(Node node) {
        if(node == null)
            return;
        // write your code here
        if(node.left == null && node.right != null)
            System.out.print(node.right.data+" ");
        if(node.right == null && node.left != null)
            System.out.print(node.left.data+" ");
        sibling(node.left);
        sibling(node.right);
    }

}
```

```
}
```

9. Binary Trees -- Print All Leaf Nodes

Given a Binary Tree Print all the leaf nodes.

Input Format

Level order input of the binary tree

Constraints

Total no of nodes <1000

Output Format

All leaf nodes from left to right in single line

Sample Input

```
1
2 3
4 5
6 7
-1 -1
-1 -1
-1 -1
-1 -1
```

Sample Output

```
4 5 6 7
```

```java
import java.util.*;
public class Main {
    public class Node {
        int val;
        Node left;
        Node right;
    }

    private Node root;

    Scanner sc = new Scanner(System.in);

    public Main() {
        // TODO Auto-generated constructor stub
        root = CreateTree();

    }

    public void Level_Order(){
        Node rootNode = this.root;
        Queue<Node> q = new LinkedList<>();
        q.add(rootNode);
        while (!q.isEmpty()) {
            Node rv = q.remove();
            if(rv.left != null){
                q.add(rv.left);
            }
            if(rv.right != null){
                q.add(rv.right);
            }
            if(rv.left == null && rv.right == null){
                System.out.print(rv.val+" ");
            }
        }
    }


    private Node CreateTree() {
        // TODO Auto-generated method stub
        int item = sc.nextInt();
        Node nn = new Node();
        nn.val = item;
        root = nn;
        Queue<Node> q = new LinkedList<>();
        q.add(nn);
        while (!q.isEmpty()) {
            Node rv = q.remove();
```

```
            int c1 = sc.nextInt();
            int c2 = sc.nextInt();
            if (c1 != -1) {
                Node ll = new Node();
                ll.val = c1;
                rv.left = ll;
                q.add(ll);

            }
            if (c2 != -1) {
                Node ll = new Node();
                ll.val = c2;
                rv.right = ll;
                q.add(ll);
            }
        }
        return root;

    }
    public static void main(String args[]) {
        Main bt = new Main();
        bt.Level_Order();
    }
}
```

10. Reverse Level Order Traversal

Given a binary tree, print it's nodes level by level in reverse order, i.e., print all nodes present at the last l
followed by nodes of the second last level and so on. Print nodes at any level from left to right.

Input Format

Enter the values of all the nodes in the binary tree in pre-order format where true suggest the node exis
false suggests it is NULL

Constraints

The number of nodes in the tree is in the range [1, 1000]
0 <= root->data <= 1000

Output Format

Reverse level order print of the node values.

Sample Input

```
10 true 20 true 40 false false true 50 false false
```

Sample Output

```
40 50 60 73 20 30 10
```

Explanation

Last level should be printed first, then second last , then third last and so on.

```java
import java.util.*;
public class Main {

    static Scanner scn = new Scanner(System.in);

    public static void main(String[] args) {
        Main m = new Main();
        BinaryTree bt1 = m.new BinaryTree();
        bt1.levelArrayList();
    }

    private class BinaryTree {
        private class Node {
            int data;
```

```
            Node left;
            Node right;
        }

        private Node root;
        private int size;

        public BinaryTree() {
            this.root = this.takeInput(null, false);
        }

        public Node takeInput(Node parent, boolean ilc) {

            int cdata = scn.nextInt();
            Node child = new Node();
            child.data = cdata;
            this.size++;

            // left
            boolean hlc = scn.nextBoolean();

            if (hlc) {
                child.left = this.takeInput(child, true);
            }

            // right
            boolean hrc = scn.nextBoolean();

            if (hrc) {
                child.right = this.takeInput(child, false);
            }

            // return
            return child;
        }

        public void levelArrayList() {
            Queue<Node>q = new LinkedList<>();
            q.add(this.root);
            ArrayList<ArrayList<Integer>> levelorder = new
ArrayList<ArrayList<Integer>>();
            while(!q.isEmpty()){
                int size = q.size();
                ArrayList<Integer>li = new ArrayList<>();
                for(int i=0;i<size;i++){
                    Node rv = q.remove();
                    if(rv.left != null){
                        q.add(rv.left);
```

```
                    }
                    if(rv.right != null){
                        q.add(rv.right);
                    }
                    li.add(rv.data);
                }
                    levelorder.add(li);
            }
            for(int i=levelorder.size()-1;i>=0;i--){
                ArrayList<Integer>li = levelorder.get(i);
                for(int j=0; j<li.size();j++){
                    System.out.print(li.get(j)+" ");
                }
            }

            // return levelorder;
            // write your code here
        }

    }

}
```

11. Largest BST in a Binary Tree

Given a Binary Tree, write a program that returns the size of the largest subtree which is also a Binc
(BST)

Input Format

The first line of input will contain an integer n. The next line will contain n integers denoting the the
traversal of the BT. The next line will contain n more integers denoting the inorder traversal of the B

Constraints

$2 \leq N \leq 10^3$

Output Format

A single integer denoting the size (no of nodes in tree) of largest BST in the given tree.

Sample Input

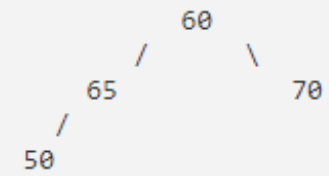
```
4
60 65 50 70
50 65 60 70
```

Sample Output

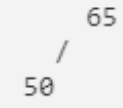
```
2
```

Explanation

The tree looks like



The largest BST subtree is



which has the size 2 i.e. it has 2 nodes in it.

```java
import java.util.Scanner;

class Main {
    static class BinaryTree {
```

```
        private class Node {
            int data;
            Node left;
            Node right;

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

        private Node root;
        private int size;

        public int size() {
            return this.size;
        }

        public boolean isempty() {
            return this.size == 0;
        }

        public BinaryTree() {
            Scanner scn = new Scanner(System.in);
            this.root = this.takeinput(scn, null, false);
        }

        private Node takeinput(Scanner scn, Node parent, boolean leftorright)
{
            if (parent == null) {
                System.out.println("Enter the data for root");
            } else {
                if (leftorright) {
                    System.out.println("Enter the data for left child of" +
parent.data);
                } else {
                    System.out.println("Enter the data for right child of" +
parent.data);
                }
            }
            int cdata = scn.nextInt();
            Node child = new Node(cdata, null, null);
            this.size++;
            boolean choice = false;
            System.out.println("Do you want have left child for" +
child.data);
            choice = scn.nextBoolean();
```

```
            if (choice) {
                child.left = this.takeinput(scn, child, true);
            }
            System.out.println("Do you have a right child");
            choice = scn.nextBoolean();
            if (choice) {
                child.right = this.takeinput(scn, child, false);
            }
            return child;
        }

        public void display() {
            this.display(this.root);
        }

        private void display(Node node) {
            if (node.left != null) {
                System.out.print(node.left.data + " =>");
            } else {
                System.out.print("END =>");
            }
            System.out.print(node.data + "<= ");
            if (node.right != null) {
                System.out.print(node.right.data);
            } else {
                System.out.print("END");
            }
            System.out.println();
            if (node.left != null) {
                this.display(node.left);
            }
            if (node.right != null) {
                this.display(node.right);
            }
        }

        ////////////////// New Constructor/////////

        public BinaryTree(int[] pre, int[] in) {
            // this.root = this.construct(pre, 0, pre.length - 1, in, 0,
in.length -
            // 1);//for preorder
            this.root = this.construct(pre, in, 0, in.length - 1);// for
                                                                  //
postorder

        }
```

```java
    private static int preIndex = 0;

    private Node construct(int[] pre, int[] in, int isi, int iei) {
        if (isi > iei) {
            return null;
        }
        Node tNode = new Node(pre[preIndex++], null, null);

        if (isi == iei) {
            return tNode;
        }

        int inIndex = search(in, isi, iei, tNode.data);
        tNode.left = construct(pre, in, isi, inIndex - 1);
        tNode.right = construct(pre, in, inIndex + 1, iei);
        return tNode;

    }

    private int search(int[] arr, int si, int ei, int data) {
        for (int i = si; i <= ei; i++) {
            if (arr[i] == data)
                return i;
        }
        return -1;
    }

    class Info {
        int size = 0;
        int max = Integer.MIN_VALUE;
        int min = Integer.MAX_VALUE;
        int ans;
        boolean isBST =true;

        Info() {

        }

        Info(int s, int max, int min, int ans, boolean isBST) {
            this.size = s;
            this.max = max;
            this.min = min;
            this.ans = ans;
            this.isBST = isBST;
        }
    }

    public int largestBSTinBT() {
```

```
            return this.largestBSTinBT(this.root).size;
        }

        //Complete this
        private Info largestBSTinBT(Node root) {
            if (root == null) {
                return new Info();
            }
            Info left = largestBSTinBT(root.left);
            Info right = largestBSTinBT(root.right);

            Info nn = new Info();
            if (left.isBST == true && right.isBST == true && left.max <
root.data && right.min > root.data) {
                nn.isBST = true;
                nn.max = Math.max(right.max, root.data);
                nn.min = Math.min(left.min, root.data);
                nn.size = left.size + right.size + 1;
                return nn;

            }

            nn.isBST = false;
            nn.max = Math.max(right.max, Math.max(left.max, root.data));
            nn.min = Math.min(right.min, Math.min(left.min, root.data));
            nn.size = Math.max(left.size, right.size);
            return nn;
        }

    }

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        int[] pre = new int[n];
        int[] in = new int[n];
        for (int i = 0; i < n; i++) {
            pre[i] = scn.nextInt();
        }
        for (int i = 0; i < n; i++) {
            in[i] = scn.nextInt();
        }

        BinaryTree bt = new BinaryTree(pre, in);
//      bt.display();
        System.out.println(bt.largestBSTinBT());
    }
}
```

12. Create tree (Using preorder and inorder)

Given preorder and inorder create the tree

Input Format

Enter the size of the preorder array N then add N more elements and store in the array denotir
traversal of the tree. Then enter the size of the inorder array M and add M more elements and
traversal of the array.

Constraints

```
1 <= N, M <= 10^4
```

Output Format

Display the tree using a modified preorder function. For each node , first print its left child's da
of the root itself , then the data of its right child. Do this for each node in a new line in preorde
the children does not exist print END in its place. Refer to the sample testcase.

Sample Input

```
3
1 2 3
3
3 2 1
```

Sample Output

```
2 => 1 <= END
3 => 2 <= END
END => 3 <= END
```

Explanation

The above tree looks like

```
          1
        /
      2
    /
```

```java
import java.util.*;
public class Main {

    static Scanner scn = new Scanner(System.in);
```

```
    public static void main(String[] args) {
        Main m = new Main();
        int[] pre = takeInput();
        int[] in = takeInput();
        BinaryTree bt = m.new BinaryTree(pre, in);
        // System.out.println(bt.root.data+" --> "+bt.root.left.data);
        bt.display();
    }

    public static int[] takeInput() {
        int n = scn.nextInt();

        int[] rv = new int[n];
        for (int i = 0; i < rv.length; i++) {
            rv[i] = scn.nextInt();
        }

        return rv;
    }

    private class BinaryTree {
        private class Node {
            int data;
            Node left;
            Node right;
        }

        private Node root;
        private int size;

        public BinaryTree(int[] pre, int[] in) {
            this.root = this.construct(pre, 0, pre.length - 1, in, 0,
in.length - 1);
        }

        private Node construct(int[] pre, int plo, int phi, int[] in, int ilo,
int ihi) {
            if(plo > phi || ilo > ihi)
                return null;

            Node root =  new Node();
            root.data = pre[plo];

            int inrootIndex = -1;
            for(int i=ilo;i<=ihi;i++){
                if(in[i] == pre[plo]){
                    inrootIndex = i;
                    break;
```

```
                }
            }

            int leftNumbers = inrootIndex - ilo;


            root.left = construct(pre, plo+1, plo+leftNumbers, in, ilo,
inrootIndex-1);

            root.right = construct(pre, plo+leftNumbers+1, phi, in,
inrootIndex+1, ihi);

            return root;
            // write your code here
        }

        public void display() {
            this.display(this.root);
        }

        private void display(Node node) {
            if (node == null) {
                return;
            }

            String str = "";

            if (node.left != null) {
                str += node.left.data;
            } else {
                str += "END";
            }

            str += " => " + node.data + " <= ";

            if (node.right != null) {
                str += node.right.data;
            } else {
                str += "END";
            }

            System.out.println(str);

            this.display(node.left);
            this.display(node.right);
        }

    }
```

```
}
```

13. Add duplicate (BST)

For each node in a binary search tree, create a new duplicate node, and insert the d
the original node.

Input Format

Enter the number of nodes N and add N more numbers to the BST

Constraints

None

Output Format

Display the tree

Sample Input

```
3
2
1
3
```

Sample Output

```
2 => 2 <= 3
1 => 2 <= END
1 => 1 <= END
END => 1 <= END
3 => 3 <=END
END => 3 <=END
```

```
import java.util.*;
public class Main {
    private class Node {
        int data;
```

```
        Node left;
        Node right;

        public Node(int data, Node left, Node right) {
            this.data = data;
            this.left = left;
            this.right = right;
        }
    }

    private Node root;
    private int size;

    public Main() {
        this.root = null;
        this.size = 0;
    }

    public int size() {
        return this.size;
    }

    public boolean isEmpty() {
        return this.size() == 0;
    }

    public void add(int data) {
        this.add(data, this.root);
    }

    private void add(int data, Node node) {
        if (this.isEmpty()) {
            Node n = new Node(data, null, null);
            this.size++;
            this.root = n;
            return;
        } else {
            if (data > node.data && node.right == null) {
                Node n = new Node(data, null, null);
                this.size++;
                node.right = n;
            } else if (data < node.data && node.left == null) {
                Node n = new Node(data, null, null);
                this.size++;
                node.left = n;
            } else if (data > node.data) {
                add(data, node.right);
            } else if (data < node.data) {
```

```
                add(data, node.left);
            }
        }
    }

    public void display() {
        this.display(this.root);
    }



    private void display(Node node) {
        if (node.left != null) {
            System.out.print(node.left.data + " => ");
        } else {
            System.out.print("END => ");
        }
        System.out.print(node.data);
        if (node.right != null) {
            System.out.print(" <= " + node.right.data);
        } else {
            System.out.print(" <= END");
        }
        System.out.println();
        if (node.left != null) {
            display(node.left);
        }
        if (node.right != null) {
            display(node.right);
        }
    }
    public void duplicate() {
            this.duplicate(this.root);
        }

        private void duplicate(Node node) {
                //Your Code Goes Here
                if(node == null)
                    return;
                Node nn = new Node(node.data,node.left,null);
                // nn.data = node.data;
                // nn.left = node.left;
                node.left = nn;

                duplicate(node.left.left);
                duplicate(node.right);
```

```
        }

        public static void main (String[] args) {
            Main Main=new Main();
            Scanner s=new Scanner(System.in);
            int n=s.nextInt();
            for(int i=0;i<n;i++){
               Main.add(s.nextInt());
            }
            Main.duplicate();
            Main.display();
        }
}
```

14. Root to Leaf (Binary Tree)

Given a binary tree and a number k, print out all root to leaf paths where the sum of
the given number.

Input Format

First line contains the values of all the nodes in the binary tree in pre-order format wh
exists and false suggests it is NULL. Second line contains the number k.

Constraints

None

Output Format

Display the root to leaf path whose sum is equal to k.

Sample Input

```
10 true 20 true 30 false false true 50 false false
60
```

Sample Output

```
10 20 30
```

Explanation

The given tree is in pre order traversal. So convert it into
binary tree and check root to leaf path sum.

```
import java.util.*;
public class Main {
    static Scanner scn = new Scanner(System.in);
    private class BinaryTree {
        private class Node {
            int data;
            Node left;
            Node right;
        }
```

```
        private Node root;
        private int size;
        public BinaryTree() {
            this.root = this.takeInput(null, false);
        }

        public Node takeInput(Node parent, boolean ilc) {

            int cdata = scn.nextInt();
            Node child = new Node();
            child.data = cdata;
            this.size++;

            // left
            boolean hlc = scn.nextBoolean();

            if (hlc) {
                child.left = this.takeInput(child, true);
            }

            // right
            boolean hrc = scn.nextBoolean();

            if (hrc) {
                child.right = this.takeInput(child, false);
            }

            // return
            return child;
        }
    }

    public static void main(String args[]) {
        Main m = new Main();
        BinaryTree bt = m.new BinaryTree();
        int k = scn.nextInt();
        // System.out.println(bt.root.data);
        pathSum(bt.root,0,"",k);
    }

    public static void pathSum(Main.BinaryTree.Node root, int sum, String
path, int k){
        if(root == null)
            return;
        if(root.left==null && root.right == null){
            if(sum+root.data == k)
                System.out.println(path+root.data);
        }
```

```
        pathSum(root.left,sum+root.data,path+root.data+" ",k);
        pathSum(root.right,sum+root.data,path+root.data+" ",k);
    }
}
```

15. Lowest Common Ancestor (Binary Tree)

Find LCA(Lowest Common Ancestor) of two elements in a Binary Tree.

Input Format

Enter the values of all the nodes in the binary tree in pre-order format where true suggest
false suggests it is NULL next two line contains 2 element of BT

Constraints

None

Output Format

Display the LCA value.

Sample Input

```
10 true 20 true 40 false false true 50 false false
50
60
```

Sample Output

```
10
```

```
import java.util.*;
public class Main {

    static Scanner scn = new Scanner(System.in);

    public static void main(String[] args) {
```

```
        Main m = new Main();
        BinaryTree bt1 = m.new BinaryTree();
        int nn1 = scn.nextInt();
        int nn2 = scn.nextInt();
        // bt1.PreOrder();
        System.out.println(LCA(bt1.root,nn1,nn2).data);
    }

    public static BinaryTree.Node LCA(BinaryTree.Node root,int nn1, int nn2){
        if(root == null)
            return null;
        if(root.data == nn1 || root.data == nn2)
            return root;

        BinaryTree.Node left = LCA(root.left,nn1,nn2);
        BinaryTree.Node right = LCA(root.right,nn1,nn2);

        if(left == null)
            return right;
        else if(right == null)
            return left;
        else
            return root;
    }



    private class BinaryTree {

        private class Node {
            int data;
            Node left;
            Node right;
        }

        private Node root;
        private int size;

        public BinaryTree() {
            this.root = this.takeInput(null, false);
        }

        public Node takeInput(Node parent, boolean ilc) {

            int cdata = scn.nextInt();
            Node child = new Node();
            child.data = cdata;
            this.size++;
```

```
            // left
            boolean hlc = scn.nextBoolean();

            if (hlc) {
                child.left = this.takeInput(child, true);
            }

            // right
            boolean hrc = scn.nextBoolean();

            if (hrc) {
                child.right = this.takeInput(child, false);
            }

            // return
            return child;
        }

    public void PreOrder() {
        PreOrder(this.root);
        System.out.println();
    }

    private void PreOrder(Node nn) {
        if(nn == null) {
            return;
        }

        System.out.print(nn.data + " ->");
        PreOrder(nn.left);
        PreOrder(nn.right);
    }

    }
}
```

16. Create Tree (Using Postorder and Inorder)

Given postorder and inorder traversal of a tree. Create the original tree on given informatio

Input Format

Enter the size of the postorder array N then add N more elements and store in the array, the inorder array M and add M more elements and store in the array. here M and N are same.

Constraints

None

Output Format

Display the tree using the display function

Sample Input

```
3
1
3
2
3
1
2
3
```

Sample Output

```
1 => 2 <= 3
END => 1 <= END
END => 3 <= END
```

```java
import java.util.*;
public class Main {

    static Scanner scn = new Scanner(System.in);

    public static void main(String[] args) {
        Main m = new Main();
        int[] post = takeInput();
        int[] in = takeInput();
        BinaryTree bt = m.new BinaryTree(post, in);
        bt.display();
    }
```

```
    public static int[] takeInput() {
        int n = scn.nextInt();

        int[] rv = new int[n];
        for (int i = 0; i < rv.length; i++) {
            rv[i] = scn.nextInt();
        }

        return rv;
    }

    private class BinaryTree {
        private class Node {
            int data;
            Node left;
            Node right;
        }

        private Node root;
        private int size;

        public BinaryTree(int[] post, int[] in) {
            this.root = this.construct(post, 0, post.length - 1, in, 0,
in.length - 1);
        }

        private Node construct(int[] post, int plo, int phi, int[] in, int
ilo, int ihi) {
            if(plo > phi || ilo > ihi)
                return null;

            Node root =  new Node();
            root.data = post[phi];

            int inrootIndex = -1;
            for(int i=ilo;i<=ihi;i++){
                if(in[i] == post[phi]){
                    inrootIndex = i;
                    break;
                }
            }

            int leftNumbers = inrootIndex - ilo;


            root.left = construct(post, plo, plo+leftNumbers-1, in, ilo,
inrootIndex-1);
```

```
            root.right = construct(post, plo+leftNumbers, phi-1,
in,inrootIndex+1, ihi );

            return root;
            // write your code here
        }

        public void display() {
            this.display(this.root);
        }

        private void display(Node node) {
            if (node == null) {
                return;
            }

            String str = "";

            if (node.left != null) {
                str += node.left.data;
            } else {
                str += "END";
            }

            str += " => " + node.data + " <= ";

            if (node.right != null) {
                str += node.right.data;
            } else {
                str += "END";
            }

            System.out.println(str);

            this.display(node.left);
            this.display(node.right);
        }

    }

}
```

17. Replace with Sum (BST)

Replace each node with the sum of all greater nodes in a given BST

Input Format

Enter the number of nodes N and add N more numbers to the BST

Constraints

None

Output Format

Display the resulting tree

Sample Input

```
3
2
1
3
```

Sample Output

```
5 => 3 <= 0
END => 5 <= END
END => 0 <= END
```

```java
import java.util.*;
public class Main {
    private class Node {
        int data;
        Node left;
        Node right;
```

```
        Node(int data, Node left, Node right) {
            this.data = data;
            this.left = left;
            this.right = right;
        }
    }

    private Node root;
    private int size;

    public Main() {
        this.root = null;
        this.size = 0;
    }

    public void add(int data) {
        if (this.isEmpty()) {
            this.root = new Node(data, null, null);
            this.size++;
        } else {
            this.add(this.root, data);
        }
    }

    private void add(Node node, int data) {
        if (data > node.data) {
            if (node.right != null) {
                this.add(node.right, data);
            } else {
                this.size++;
                node.right = new Node(data, null, null);
            }
        } else if (data < node.data) {
            if (node.left != null) {
                this.add(node.left, data);
            } else {
                this.size++;
                node.left = new Node(data, null, null);
            }
        } else {
            // nothing to do
        }
    }

    public void remove(int data) {
        this.root = this.remove(this.root, data);
    }
```

```java
    private Node remove(Node node, int data) {
        if (node == null) {
            return null;
        }

        if (data > node.data) {
            node.right = this.remove(node.right, data);
            return node;
        } else if (data < node.data) {
            node.left = this.remove(node.left, data);
            return node;
        } else {
            // found the element
            if (node.left == null && node.right == null) {
                this.size--;
                return null;
            } else if (node.left == null) {
                this.size--;
                return node.right;
            } else if (node.right == null) {
                this.size--;
                return node.left;
            } else {
                // both children
                int lmax = this.max(node.left);
                node.data = lmax;
                node.left = this.remove(node.left, lmax);

                return node;
            }
        }
    }

    public int size() {
        return this.size;
    }

    public boolean isEmpty() {
        return this.size() == 0;
    }

    public void display() {
        System.out.println(this);
    }

    public int max() {
        return this.max(this.root);
    }
```

```
    private int max(Node node) {
        int rv = node.data;

        if (node.right != null) {
            rv = this.max(node.right);
        }

        return rv;
    }

    @Override
    public String toString() {
        return this.toString(this.root);
    }

    private String toString(Node node) {
        if (node == null) {
            return "";
        }

        String retVal = "";

        if (node.left != null) {
            retVal += node.left.data + " => ";
        } else {
            retVal += "END" + " => ";
        }

        retVal += node.data;

        if (node.right != null) {
            retVal += " <= " + node.right.data;
        } else {
            retVal += " <= " + "END";
        }

        retVal += "\n";

        retVal += this.toString(node.left);
        retVal += this.toString(node.right);

        return retVal;
    }
    private int sum;
public void replaceWLS(Node root) {
        if(root == null)
            return;
```

```
        replaceWLS(root.right);
        int riData = root.data;
        root.data = sum;
        sum+=riData;
        replaceWLS(root.left);
    }
public static void main(String[] args) {
        Main b1= new Main();
        Scanner scn = new Scanner(System.in);
        int n= scn.nextInt();
        while(n!=0){
            int m=scn.nextInt();
            b1.add(m);
            --n;
        }
 b1.replaceWLS(b1.root);
System.out.println(b1);
}
}
```

18. Binary Tree - Max Path Sum

Given a binary tree , print its max path sum.
For this problem, a path is defined as any sequence of nodes from some starting node to ar
along the parent-child connections. The path must contain at least one node and does not
the root.

Input Format

Single line input containing space separated integers denoting the preorder input of the tre
the node does not exist.

Constraints

1 <= No of nodes <= 10^5

Output Format

Print a single integer denoting the max path sum for the given tree.

Sample Input

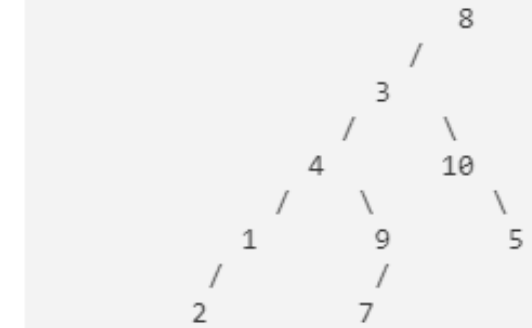
```
8 3 4 1 2 NULL NULL NULL 9 7 NULL NULL NULL 10 NULL
```

Sample Output

```
38
```

Explanation

The tree looks like



The max path sum is given by : 7 + 9 + 4 + 3 + 10 + 5 = 38

```java
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;

class Main{
    public class Node{
        int val;
        Node left;
        Node right;
    }
    private Node root;
    public Main(){
         root=CreateTree();
    }
    static Scanner pscn=new Scanner(System.in);
    private Node CreateTree() {
        String item = pscn.next();
        if (item.equals("NULL")) {
            return null;
        }

        Node nn = new Node();
        nn.val = Integer.parseInt(item);
        nn.left = CreateTree();
        nn.right = CreateTree();

        return nn;
    }

    int MaxPath=Integer.MIN_VALUE;
    int max_path(Node root){
        if(root==null) return 0;
        int left=Math.max(max_path(root.left),0);
        int right=Math.max(max_path(root.right),0);
        MaxPath=Math.max(MaxPath,left+right+root.val);
        return Math.max(left,right)+root.val;
    }

    public static void main(String[] args) {
        Main m=new Main();
        int a=m.max_path(m.root);
        System.out.println(m.MaxPath);
    }
}
```

19. Root to Leaf - ||

Given a binary tree and an integer sum,Print count of root-to-leaf paths that have path's sum equal to integer sum.

Input Format

The first line contains level order traversal of the tree . In the level order traversal given, -1 represent a nu while any other value represent a node of the tree.
Next line contains a single integer sum.

Constraints

1<=number of nodes in the tree <=10^5

Output Format

Print a single integer equal to number of root to leaf paths with given sum.

Sample Input

```
1 2 3 -1 -1 -1 -1
3
```

Sample Output

```
1
```

Explanation

There is only 1 root to leaf path with sum 3 (from 1 to 2).

```java
import java.util.*;
public class Main {

        public class Node {
        int val;
        Node left;
        Node right;
    }
```

```
    private Node root;
    private int sum;

    Scanner sc = new Scanner(System.in);

    public Main() {
        // TODO Auto-generated constructor stub
        root = CreateTree();
        sum = sc.nextInt();
        // System.out.println(sum);
    }


    private Node CreateTree() {
        // TODO Auto-generated method stub
        int item = sc.nextInt();
        Node nn = new Node();
        nn.val = item;
        root = nn;
        Queue<Node> q = new LinkedList<>();
        q.add(nn);
        while (!q.isEmpty()) {
            Node rv = q.remove();
            int c1 = sc.nextInt();
            int c2 = sc.nextInt();
            if (c1 != -1) {
                Node ll = new Node();
                ll.val = c1;
                rv.left = ll;
                q.add(ll);

            }
            if (c2 != -1) {
                Node ll = new Node();
                ll.val = c2;
                rv.right = ll;
                q.add(ll);
            }
            // System.out.println(c1+" "+c2);
        }
        // int n = sc.nextInt();
        // System.out.println(n);
        return root;

    }
    public static void main(String args[]) {
        Main bt = new Main();
        // Scanner sc = new Scanner(System.in);
```

```
        int sum = bt.sum;
        System.out.println(pathCount(sum,bt.root));
        // System.out.println(bt.root.val);
    }

    public static int pathCount(int sum, Main.Node root){
        if(root == null)
            return 0;
        if(root.left == null && root.right == null && sum-root.val == 0)
            return 1;

        int count = pathCount(sum-root.val,root.left)+pathCount(sum-
root.val,root.right);
        // System.out.println(0);
        return count;
    }
}
```

20. Mirror Tree

Given a binary tree , Check if its mirror image is equal to the tree itself.

Input Format

The first line of the input contains level order traversal of the tree as space separated inte
traversal , -1 represent a null child while any other value represent a node of the tree.

Constraints

1<=number of nodes in the tree <=10^5

Output Format

Print "YES" without quotes if the mirror image of the tree is equal to the tree itself , else pri

Sample Input

```
1 2 2 -1 -1 -1 -1
```

Sample Output

```
YES
```

Explanation

The mirror image of given tree is equal to the tree itself.

```java
import java.util.*;
public class Main {
 public class Node {
        int val;
        Node left;
        Node right;
    }

    private Node root;

    Scanner sc = new Scanner(System.in);

    public Main() {
```

```
        // TODO Auto-generated constructor stub
        root = CreateTree();
        // System.out.println(sum);
    }


    private Node CreateTree() {
        // TODO Auto-generated method stub
        int item = sc.nextInt();
        Node nn = new Node();
        nn.val = item;
        root = nn;
        Queue<Node> q = new LinkedList<>();
        q.add(nn);
        while (!q.isEmpty()) {
            Node rv = q.remove();
            int c1 = sc.nextInt();
            int c2 = sc.nextInt();
            if (c1 != -1) {
                Node ll = new Node();
                ll.val = c1;
                rv.left = ll;
                q.add(ll);

            }
            if (c2 != -1) {
                Node ll = new Node();
                ll.val = c2;
                rv.right = ll;
                q.add(ll);
            }
        }
        return root;

    }
    public static void main(String args[]) {
        Main bt = new Main();
        if(bt.Mirror(bt.root.left,bt.root.right))
            System.out.println("YES");
        else
            System.out.println("NO");

    }
    public boolean Mirror(Main.Node root1, Main.Node root2) {
            if (root1 == null && root2 == null) {
                return true;
            }
            if (root1 == null || root2 == null) {
                return false;
```

```
            }
            if (root1.val != root2.val) {
                return false;
            }

            boolean left = Mirror(root1.left, root2.right);
            boolean right = Mirror(root1.right, root2.left);
            return left && right;

        }
}
```

21. Replace with Sum of greater nodes

Given a binary search tree, replace each nodes' data with the sum of all nodes' which are greater
it. Include the current node's data also.

Input Format

The first line contains a number n showing the length of the inorder array of BST. The next line cont
denoting the elements of the array.

Constraints

$2 \leq N \leq 10^3$

Output Format

Print the preorder traversal of the new tree.

Sample Input

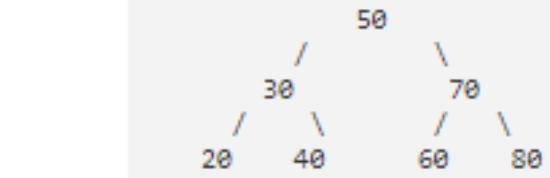
```
7
20
30
40
50
60
70
80
```

Sample Output

```
260 330 350 300 150 210 80
```

Explanation

The original tree looks like



We are supposed to replace the elements by the sum of elements larger than it.
80 being the largest element remains unaffected .
70 being the second largest element gets updated to 150 (70+80)
60 becomes 210 (60 + 70 + 80)
50 becomes 260 (50 + 60 + 70 + 80)
40 becomes 300 (40 + 50 + 60 + 70 + 80)
30 becomes 330 (30 + 40 + 50 + 60 + 70 + 80)
20 becomes 350 (20 + 30 + 40 + 50 + 60 + 70 + 80)

The new tree looks like

```java
import java.util.*;
public class Main {
    public class Node{
        int val;
        Node left;
        Node right;
    }
    Scanner Pscn = new Scanner(System.in);
    private Node root;

    public Main(int []arr) {
        root = createTree(arr,0,arr.length-1);
    }
    private Node createTree(int []arr, int si, int ei) {
        if(si > ei)
            return null;
        int mid = si+(ei-si)/2;
        Node nn = new Node();
        nn.val = arr[mid];
        nn.left = createTree(arr,si,mid-1);
        nn.right = createTree(arr,mid+1,ei);
        return nn;
    }
    private int sum;
    public void replace(Node root){
        if(root == null)
            return;
        replace(root.right);
        int rpData = root.val;
        root.val += sum;
        sum+=rpData;
        replace(root.left);
    }

    public void PreOrder(Node nn) {
        if(nn == null) {
            return;
        }

        System.out.print(nn.val + " ");
        PreOrder(nn.left);
        PreOrder(nn.right);
    }
    public static void main(String args[]) {
        Scanner pscn = new Scanner(System.in);
        int n = pscn.nextInt();
```

```
        int []arr = new int [n];
        for(int i=0; i<n;i++){
            arr[i] = pscn.nextInt();
        }
        Main tr = new Main(arr);
        tr.replace(tr.root);
        tr.PreOrder(tr.root);
    }
}
```

22. Binary Tree Maximum Path Sum

Given a binary tree, find the maximum path sum in it.
The path is defined as a sequence of nodes that follows parent-child connection.Path may start from any noc
and end at any node.

Input Format

The first line of input contains level order traversal of the tree.In the input , a -1 value represent a null child while
any other value represent a node in the tree.

Constraints

1<=number of nodes in the tree <=10^5

Output Format

Print a single integer equal to maximum path sum in the given tree.

Sample Input

```
1 4 6 -1 -1 -1 -1
```

Sample Output

```
11
```

Explanation

The path with maximum sum is from the root's left child(4) to root's right child(6) with sum as 11(4+1+6)

```java
import java.util.*;
public class Main {

    public class Node {
        int val;
        Node left;
        Node right;
    }

    private Node root;

    Scanner sc = new Scanner(System.in);

    public Main() {
        // TODO Auto-generated constructor stub
        root = CreateTree();

    }


    private Node CreateTree() {
        // TODO Auto-generated method stub
        Node nn = new Node();
        root = nn;
        Queue<Node> q = new LinkedList<>();
        nn.val = sc.nextInt();
        q.add(nn);
        while(!q.isEmpty()) {
            Node rv = q.remove();
            int c1 = sc.nextInt();
            if (c1 != -1) {
                // System.out.println(arr[i]);
                Node ll = new Node();
                ll.val = c1;
                rv.left = ll;
                q.add(ll);
            }

            int c2 = sc.nextInt();
            if (c2 != -1) {
                // System.out.println(arr[i]);
                Node ll = new Node();
                ll.val = c2;
                rv.right = ll;
                q.add(ll);
            }
```

```
        }
        return root;

    }
    public static void main(String args[]) {
        Main bt = new Main();
        // bt.Level_Order();
        // System.out.println(bt.MaxDepth(bt.root.left,0));
        bt.MaxPath(bt.root);
        System.out.println(maxPath);

    }
    static int maxPath = Integer.MIN_VALUE;
    public static int MaxPath(Main.Node root){
        if(root == null){
            return 0;
        }

        int lm = Math.max(0,MaxPath(root.left));
        int rm = Math.max(0,MaxPath(root.right));

        maxPath = Math.max(maxPath, lm+rm+root.val);
        return Math.max(lm,rm)+root.val;
    }
}
```

23. Maximum Depth of Binary Tree

Given a binary tree, find the maximum depth of the tree.

Input Format

Enter the value of the node then the Boolean choice i.e whether the node has a left child , then enter the child's data . The input acts in a recursive manner i.e when the left child's children are made only then we onto the parent's right child

Constraints

None

Output Format

Print the depth of the binary tree.

Sample Input

```
10 true 20 true 73 false false true 50 false false
```

Sample Output

```
3
```

Explanation

Calculate the maximum height of the tree.

```java
import java.util.*;
public class Main {

static Scanner scn = new Scanner(System.in);
    private class BinaryTree {
        private class Node {
            int data;
            Node left;
            Node right;
        }

        private Node root;
        private int size;
        public BinaryTree() {
            this.root = this.takeInput(null, false);
        }
```

```
        public Node takeInput(Node parent, boolean ilc) {

            int cdata = scn.nextInt();
            Node child = new Node();
            child.data = cdata;
            this.size++;

            // left
            boolean hlc = scn.nextBoolean();

            if (hlc) {
                child.left = this.takeInput(child, true);
            }

            // right
            boolean hrc = scn.nextBoolean();

            if (hrc) {
                child.right = this.takeInput(child, false);
            }

            // return
            return child;
        }
    }

    public static void main(String args[]) {
        Main m = new Main();
        BinaryTree bt = m.new BinaryTree();
        MaxDepth(bt.root,0);
        System.out.print(maxDepth);
    }

    static int maxDepth = 0;
    public static void MaxDepth(Main.BinaryTree.Node root, int depth){
        if(root == null){
            maxDepth = Math.max(depth,maxDepth);
            return;
        }
        MaxDepth(root.left,depth+1);
        MaxDepth(root.right,depth+1);
    }
}
```

24. Determine whether a given binary tree is a BST or not

Given a binary tree, determine whether it is a BST.

Input Format

Level order traversals given where -1 means child is NULL

Constraints

-10^5<=nodes.data<=10^5

Output Format

true or false

Sample Input

```
20 10 30 -1 -1 5 40 -1 -1 -1 -1
```

Sample Output

```
false
```

Explanation

In the tree above, each node meets the condition that the node contains a value larger than its left child and smaller than its right child hold, and yet it's not a BST: the value 5 is on the right subtree of the node containing 20, a violation of the BST property.

```
import java.util.LinkedList;
```

```java
import java.util.Queue;
import java.util.Scanner;

class Main {
    static class BinaryTreeFromLvlOrder {

        private Scanner s = new Scanner(System.in);

        private class Node {
            int val;
            Node left;
            Node right;

            public Node(int val) {
                this.val = val;
            }
        }

        private Node root;

        public BinaryTreeFromLvlOrder() {
            // TODO Auto-generated constructor stub
            root = construct();// 2k
        }

        private Node construct() {
            // TODO Auto-generated method stub

            int val = s.nextInt();
            Node nn = new Node(val);
            Queue<Node> lvl = new LinkedList<>();
            lvl.add(nn);

            while (!lvl.isEmpty()) {
                Node front = lvl.remove();
                int a = s.nextInt();
                int b = s.nextInt();

                if (a != -1) {
                    Node na = new Node(a);
                    front.left = na;
                    lvl.add(na);
                }

                if (b != -1) {
                    Node nb = new Node(b);
                    front.right = nb;
                    lvl.add(nb);
```

```
                }
            }
            return nn;// 2k
        }

        public class BstPair {
            boolean isbst = true;
            long min = Long.MAX_VALUE;
            long max = Long.MIN_VALUE;
        }

        private boolean isValid(Node root) {
            return validbst(root).isbst;
        }

        public BstPair validbst(Node root) {
            if (root == null) {
                return new BstPair();
            }

            BstPair left = validbst(root.left);
            BstPair right = validbst(root.right);

            BstPair sb = new BstPair();
            if (left.isbst == true && right.isbst == true && left.max <
root.val && right.min > root.val) {
                sb.isbst = true;
                sb.max = Math.max(right.max, root.val);
                sb.min = Math.min(left.min, root.val);
                return sb;

            }

            sb.isbst = false;
            sb.max = Math.max(right.max, Math.max(left.max, root.val));
            sb.min = Math.min(right.min, Math.min(left.min, root.val));
            return sb;

        }

    }

    public static void main(String[] args) {
        BinaryTreeFromLvlOrder bt = new BinaryTreeFromLvlOrder();
        System.out.println(bt.isValid(bt.root));
    }
}
```

25. Bottom Up Level Order

Given a Binary tree, print the *bottom-up level order traversal* of its nodes, i.e. from left to right, eac to root.

Input Format

The only line of input contains N space-separated values, i.e. integers or N denoting a null node in order traversal.

Constraints

$|node.Value| \leq 10^5$

Output Format

Print each level on a new line, where the values of each node in the level are separated by a spac

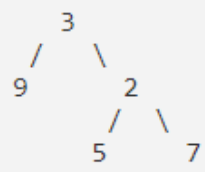
Sample Input

```
3 9 2 N N 5 7
```

Sample Output

```
5 7
9 2
3
```

Explanation



The levels in bottom-up would be, {5, 7}, {9, 2}, {3}.

```java
import java.util.*;

class TreeNode {
    int value;
    TreeNode left;
    TreeNode right;

    public TreeNode(int value) {
        this.value = value;
        this.left = null;
        this.right = null;
    }
}

public class Main {
    public static void pBTN(TreeNode root) {
        if (root == null) {
            return;
        }

        List<List<Integer>> result = new ArrayList<>();
        Queue<TreeNode> queue = new LinkedList<>();
        queue.offer(root);

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

            for (int i = 0; i < lvlSize; i++) {
                TreeNode node = queue.poll();
                lvlValues.add(node.value);

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

            result.add(0, lvlValues);
        }

        for (List<Integer> lvl : result) {
            for (int value : lvl) {
                System.out.print(value + " ");
```

```
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        Scanner pscn = new Scanner(System.in);
        String[] elmt = pscn.nextLine().split(" ");
        TreeNode root = new TreeNode(Integer.parseInt(elmt[0]));
        Queue<TreeNode> queue = new LinkedList<>();
        queue.offer(root);
        int idx = 1;

        while (!queue.isEmpty() && idx < elmt.length) {
            TreeNode node = queue.poll();
            if (!elmt[idx].equals("N")) {
                node.left = new TreeNode(Integer.parseInt(elmt[idx]));
                queue.offer(node.left);
            }
            idx++;

            if (idx < elmt.length && !elmt[idx].equals("N")) {
                node.right = new TreeNode(Integer.parseInt(elmt[idx]));
                queue.offer(node.right);
            }
            idx++;
        }

        pBTN(root);
    }
}
```