Binary Tree In-order Traversal

1. Binary Tree Inorder Traversal

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
       public List<Integer> res = new ArrayList<>();
    public List<Integer> inorderTraversal(TreeNode root) {

        // recursiveInorderTraversalUtil(root);
        iterativeInorderTraversalUtil(root);
        return res;
    }

    private void recursiveInorderTraversalUtil(TreeNode root){

        // base case
        if (root == null){
            return;
        }
        recursiveInorderTraversalUtil(root.left);
        res.add(root.val);
        recursiveInorderTraversalUtil(root.right);
    }

    private void iterativeInorderTraversalUtil(TreeNode root){

        // base case
        if (root == null) return;
```

```
        Stack<TreeNode> st = new Stack<>();

        TreeNode node = root;
        while (true){
            if (node != null){
                st.push(node);
                node = node.left;
            }
            else{
                if (st.isEmpty()){
                    break;
                }
                node = st.pop();
                res.add(node.val);
                node = node.right;
            }
        }
    }
}
```

Binary Tree Preorder Traversal

https://leetcode.com/problems/binary-tree-level-order-traversal/description/

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {

        Queue<TreeNode> queue = new LinkedList<>();
```

```
        List<List<Integer>> wrapList = new ArrayList<>();

        if (root == null) return wrapList;

        queue.offer(root);

        while (!queue.isEmpty()){
            int levelNum = queue.size();
            List<Integer> subList = new ArrayList<>();
            for (int i = 0; i < levelNum; i++){
                if (queue.peek().left != null) queue.offer(queue.peek().left);
                if (queue.peek().right != null)
queue.offer(queue.peek().right);
                subList.add(queue.poll().val);
            }
            wrapList.add(subList);
        }

        return wrapList;
    }
}
```

Binary Tree Post-order Traversal

https://leetcode.com/problems/binary-tree-postorder-traversal/description/

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public List<Integer> res;
    public List<Integer> postorderTraversal(TreeNode root) {
```

```
        res = new ArrayList<>();
        recursivePostorderTraversalUtil(root);
        // iterativePostorderTraversalUtil(root);

        return res;

    }

    private void recursivePostorderTraversalUtil(TreeNode root){

        // base case
        if (root == null) return;

        recursivePostorderTraversalUtil(root.left);
        recursivePostorderTraversalUtil(root.right);
        res.add(root.val);
    }

    private void  iterativePostorderTraversalUtil(TreeNode root){

        // base case
        if (root == null) return;

        Stack<TreeNode> st1 = new Stack<>();
        Stack<TreeNode> st2 = new Stack<>();

        st1.push(root);

        while (!st1.isEmpty()){
            root = st1.pop();
            st2.push(root);
            if (root.left != null) st1.push(root.left);
            if (root.right != null) st1.push(root.right);
        }

        while (!st2.isEmpty()){
            res.add(st2.pop().val);
        }
    }
}
```

Binary Tree Level Order Traversal

https://leetcode.com/problems/binary-tree-level-order-traversal/description/

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {

        Queue<TreeNode> queue = new LinkedList<>();
        List<List<Integer>> wrapList = new ArrayList<>();

        if (root == null) return wrapList;

        queue.offer(root);

        while (!queue.isEmpty()){
            int levelNum = queue.size();
            List<Integer> subList = new ArrayList<>();
            for (int i = 0; i < levelNum; i++){
                if (queue.peek().left != null) queue.offer(queue.peek().left);
                if (queue.peek().right != null)
queue.offer(queue.peek().right);
                subList.add(queue.poll().val);
            }
            wrapList.add(subList);
        }

        return wrapList;
    }
}
```

Maximum Depth of Binary Tree

https://leetcode.com/problems/maximum-depth-of-binary-tree/description/

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public int maxDepth(TreeNode root) {

        return maxDepthUtil(root);
    }

    private int maxDepthUtil(TreeNode root){

        // base case
        if (root == null) return 0;

        int leftHeight = maxDepthUtil(root.left);
        int rightHeight = maxDepthUtil(root.right);

        return 1 + Math.max(leftHeight, rightHeight);
    }
}
```

Balanced Binary Tree

https://leetcode.com/problems/balanced-binary-tree/description/

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
```

```
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public boolean isBalanced(TreeNode root) {

        return dfsHeight(root) != -1;
    }

    private int dfsHeight(TreeNode root){

        // base case
        if (root == null) return 0;

        int leftHeight = dfsHeight(root.left);
        if (leftHeight == -1) return -1;

        int rightHeight = dfsHeight(root.right);
        if (rightHeight == -1) return -1;

        if (Math.abs(rightHeight - leftHeight) > 1) return -1;

        return 1 + Math.max(leftHeight, rightHeight);
    }
}
```

Diameter of a Binary Tree

https://leetcode.com/problems/diameter-of-binary-tree/description/

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
```

```
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    int maxi;
    public int diameterOfBinaryTree(TreeNode root) {

        maxi = Integer.MIN_VALUE;
        findHeight(root);
        return maxi;
    }

    private int findHeight(TreeNode root){
        if (root == null){
            return 0;
        }
        int leftHeight = findHeight(root.left);
        int rightHeight = findHeight(root.right);
        maxi = Math.max(maxi,leftHeight + rightHeight);
        return 1 + Math.max(leftHeight,rightHeight);
    }
}
```

Maximum Path Sum in Binary Tree

https://leetcode.com/problems/binary-tree-maximum-path-sum/description/

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
```

```
 */
class Solution {
    int maxi;
    public int maxPathSum(TreeNode root) {
        maxi = Integer.MIN_VALUE;
        findMaxPathSum(root);
        return maxi;
    }

    private int findMaxPathSum(TreeNode root){
        // base case
        if (root == null){
            return 0;
        }
        int left = Math.max(0,findMaxPathSum(root.left));
        int right = Math.max(0,findMaxPathSum(root.right));
        maxi = Math.max(maxi, left + right + root.val);

        return root.val + Math.max(left, right);
    }
}
```

Check if two Trees are identical or Not

```
Do any traversal, if you get the same output, then true else false.
```

https://leetcode.com/problems/same-tree/description/

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
```

```java
class Solution {
    public boolean isSameTree(TreeNode p, TreeNode q) {
        return isSameTreeUtil(p,q);
    }

    private boolean isSameTreeUtil(TreeNode p, TreeNode q){
        if (p == null || q == null){
            return (p == q);
        }

        boolean left = isSameTreeUtil(p.left,q.left);
        boolean right = isSameTreeUtil(p.right,q.right);

        return (p.val == q.val) && left && right;

    }
}
```

Binary Tree Zigzag Level Order Traversal

https://leetcode.com/problems/binary-tree-zigzag-level-order-traversal/description/

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public List<List<Integer>> zigzagLevelOrder(TreeNode root) {

        Queue<TreeNode> q = new LinkedList<>();
        List<List<Integer>> wrapList = new ArrayList<>();

        if (root == null) return wrapList;
```

```
        q.offer(root);
        boolean leftToRight = true;

        while (!q.isEmpty()){
            int levelNum = q.size();
            List<Integer> subList = new ArrayList<>();
            for (int i = 0; i< levelNum; i++){
                if (q.peek().left != null) q.offer(q.peek().left);
                if (q.peek().right != null) q.offer(q.peek().right);

                subList.add(q.poll().val);
            }
            if (!leftToRight){
                Collections.reverse(subList);
            }
            wrapList.add(subList);
            leftToRight = !leftToRight;
        }

        return wrapList;

    }
}
```

Boundary Traversal in Binary Tree (Anti-Clock wise)

1. Add left boundary exclusive leaf nodes
2. Add leaf nodes
3. Add right boundary in reverse order exclusive leaf nodes

https://www.geeksforgeeks.org/problems/boundary-traversal-of-binary-tree/1

```
//User function Template for Java

// class Node
// {
//     int data;
//     Node left, right;

//     public Node(int d)
//     {
//         data = d;
```

```
//         left = right = null;
//     }
// }

class Solution
{
        ArrayList <Integer> boundary(Node root)
        {
            ArrayList<Integer> res = new ArrayList<>();

            if (root == null) return res;

            if (isLeaf(root) == false) res.add(root.data);

            addLeftBoundaryUtil(root,res);
            addLeavesUtil(root,res);
            addRightBoundaryUtil(root,res);

            return res;
        }

        private void addLeftBoundaryUtil(Node root, ArrayList<Integer> res){
            Node curr = root.left;
            while (curr != null){
                if (isLeaf(curr) == false) res.add(curr.data);
                if (curr.left != null) curr = curr.left;
                else curr = curr.right;
            }
        }

        private void addLeavesUtil(Node root, ArrayList<Integer> res){
            if (isLeaf(root)){
                res.add(root.data);
                return;
            }
            if (root.left != null) addLeavesUtil(root.left, res);
            if (root.right != null) addLeavesUtil(root.right, res);
        }

        private void addRightBoundaryUtil(Node root, ArrayList<Integer> res){
            Node curr = root.right;
            ArrayList<Integer> tmp = new ArrayList<>();
            while (curr != null){
                if (isLeaf(curr) == false) tmp.add(curr.data);
                if (curr.right != null) curr = curr.right;
                else curr = curr.left;
```

```
            }
            for (int i = tmp.size()-1; i >=0; i--){
                res.add(tmp.get(i));
            }
        }

        private boolean isLeaf(Node root){
            if (root.left == null && root.right == null) return true;
            else return false;
        }
}
```

Vertical Order Traversal of Binary Tree

```
- If 2 nodes are overlaping then write in the sort order
```

- https://leetcode.com/problems/vertical-order-traversal-of-a-binary-tree/description/

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */

 class Tuple{
     TreeNode node;
     int row;
     int col;
     public Tuple(TreeNode _node, int _row, int _col){
         node = _node;
         row = _row;
         col = _col;
     }
```

```
 }
class Solution {
    public List<List<Integer>> verticalTraversal(TreeNode root) {

        TreeMap<Integer,TreeMap<Integer,PriorityQueue<Integer>>> map = new
TreeMap<>();
        Queue<Tuple> q = new LinkedList<>();
        q.offer(new Tuple(root, 0, 0));

        while (!q.isEmpty()){
            Tuple tuple = q.poll();
            TreeNode node = tuple.node;
            int x = tuple.row;
            int y = tuple.col;

            if (!map.containsKey(x)){
                map.put(x, new TreeMap<>());
            }

            if (!map.get(x).containsKey(y)){
                map.get(x).put(y, new PriorityQueue<>());
            }

            map.get(x).get(y).offer(node.val);

            if (node.left != null){
                q.offer(new Tuple(node.left, x-1, y+1));
            }
            if (node.right != null){
                q.offer(new Tuple(node.right, x+1, y+1));
            }
        }

        List<List<Integer>> list = new ArrayList<>();
        for (TreeMap<Integer, PriorityQueue<Integer>> ys : map.values()){
            list.add(new ArrayList<>());
            for (PriorityQueue<Integer> nodes : ys.values()){
                while (!nodes.isEmpty()){
                    list.get(list.size()-1).add(nodes.poll());
                }
            }
        }

        return list;
    }
}
```

Top View of Binary Tree

https://www.geeksforgeeks.org/problems/top-view-of-binary-tree/1

```
/*
class Node{
    int data;
    Node left;
    Node right;
    Node(int data){
        this.data = data;
        left=null;
        right=null;
    }
}
*/
class Pair{
    Node node;
    int hd;
    public Pair(Node _node, int _hd){
        node = _node;
        hd = _hd;
    }
}

class Solution
{
    //Function to return a list of nodes visible from the top view
    //from left to right in Binary Tree.
    static ArrayList<Integer> topView(Node root)
    {
        // add your code
        ArrayList<Integer> ans = new ArrayList<>();
        if (root == null) return ans;

        Map<Integer, Integer> map = new TreeMap<>();
        Queue<Pair> q =new LinkedList<Pair>();
        q.add(new Pair(root,0));

        while (!q.isEmpty()){
            Pair it = q.remove();
            int hd = it.hd;
            Node temp = it.node;

            if (map.get(hd)==null) map.put(hd,temp.data);
```

```
            if (temp.left != null){
                q.add(new Pair(temp.left,hd-1));
            }

            if (temp.right != null){
                q.add(new Pair(temp.right, hd+1));
            }
        }

        for (Map.Entry<Integer,Integer> entry : map.entrySet()){
            ans.add(entry.getValue());
        }
        return ans;
    }
}
```

Bottom View of the Binary Tree

1. If there are two nodes at the same place then always take the right one
2. Print the last node of vertical order tarversal

https://www.geeksforgeeks.org/problems/bottom-view-of-binary-tree/1

```
//User function Template for Java


class Solution
{
    //Function to return a list containing the bottom view of the given tree.
    public ArrayList <Integer> bottomView(Node root)
    {
        // Code here
        ArrayList<Integer> res = new ArrayList<>();

        if (root == null) return res;

        Map<Integer,Integer> map = new TreeMap<>();
        Queue<Node> q = new LinkedList<>();
        root.hd = 0; // horizontal distance of the node;

        q.add(root);
        while (!q.isEmpty()){
            Node temp = q.remove();
```

```
            int hd = temp.hd;
            map.put(hd,temp.data);

            if (temp.left != null){
                temp.left.hd = hd-1;
                q.add(temp.left);
            }
            if (temp.right != null){
                temp.right.hd = hd + 1;
                q.add(temp.right);
            }
        }

        for (Map.Entry<Integer,Integer> it : map.entrySet()){
            res.add(it.getValue());
        }

        return res;

    }
}
```

Right/Left View of a Binary Tree

Right Side View

https://leetcode.com/problems/binary-tree-right-side-view/description/

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
```

```
    public List<Integer> res;
    public List<Integer> rightSideView(TreeNode root) {
        res = new ArrayList<>();
        rightSideViewUtil(root, 0);
        return res;
    }

    private void rightSideViewUtil(TreeNode root, int level){

        // base case
        if (root == null){
            return;
        }

        if (level == res.size()){
            res.add(root.val);
        }
        rightSideViewUtil(root.right, level+1);
        rightSideViewUtil(root.left, level+1);
    }
}
```

Left Side View

https://www.geeksforgeeks.org/problems/left-view-of-binary-tree/1

```
//User function Template for Java

/* A Binary Tree node
class Node
{
    int data;
    Node left, right;

    Node(int item)
    {
        data = item;
        left = right = null;
    }
}*/
class Tree
{
    public ArrayList<Integer> res;
    //Function to return list containing elements of left view of binary tree.
```

```
    ArrayList<Integer> leftView(Node root)
    {
      // Your code here
      res = new ArrayList<>();
      leftSideViewUtil(root, 0);
      return res;
    }

    private void leftSideViewUtil(Node root, int level){

        // base case
        if (root == null){
            return;
        }

        if (res.size() == level){
            res.add(root.data);
        }

        leftSideViewUtil(root.left, level + 1);
        leftSideViewUtil(root.right, level + 1);
    }
}
```

Check for Symmetrical Binary Trees

https://leetcode.com/problems/symmetric-tree/

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
```

```
    public boolean isSymmetric(TreeNode root) {
        return root == null || isSymmetricUtil(root.left, root.right);
    }

    private boolean isSymmetricUtil(TreeNode left, TreeNode right){

        // base case
        if (left == null || right == null){
            return left == right;
        }

        if (left.val != right.val) return false;

        return isSymmetricUtil(left.left, right.right) &&
isSymmetricUtil(left.right, right.left);
    }
}
```

Print Root to Node Path in Binary Tree

https://leetcode.com/problems/binary-tree-paths/description/

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    List<String> res;
    List<Integer> res1;
    public List<String> binaryTreePaths(TreeNode root) {
        res = new ArrayList<>();
        res1 = new ArrayList<>();
        if (root != null){
```

```
            binaryTreePathsUtil(root, "");
        }
        return res;
    }

    private void binaryTreePathsUtil(TreeNode root, String s){
        // base case
        if (root == null) return;

        if (s.isEmpty()){
            s += root.val;
        }
        else s += ("->"+root.val);

        if (root.left != null || root.right != null){
            binaryTreePathsUtil(root.left, s);
            binaryTreePathsUtil(root.right, s);
        }

        else{
            res.add(s);
        }
    }

    private boolean getPath(TreeNode root, int ele){

        // base case
        if (root == null) return false;

        res1.add(root.val);

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

        if (getPath(root.left, ele) || getPath(root.right, ele)) return true;

        // if not found on this path then remove the last added value and
return false
        res1.remove(res1.size()-1);

        return false;
    }
}
```

Lowest Common Ancestor of a Binary Tree

https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/description/

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode
q) {

        return lowestCommonAncestorUtil(root, p, q);
    }

    private TreeNode lowestCommonAncestorUtil(TreeNode root, TreeNode p,
TreeNode q){

        // base case
        if (root == null || root == p || root ==q) return root;

        TreeNode left = lowestCommonAncestorUtil(root.left, p, q);
        TreeNode right = lowestCommonAncestorUtil(root.right, p, q);

        if (left == null) return right;

        else if (right == null) return left;

        else { // both left and right are not null , we found are result
            return root;
        }
    }
}
```

Maximum Width of a Binary Tree

https://leetcode.com/problems/maximum-width-of-binary-tree/description/

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
```

```
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
 class Pair{
     TreeNode node;
     int num;
     Pair(TreeNode _node, int _num){
        node = _node;
        num = _num;
     }
 }
class Solution {
    public int widthOfBinaryTree(TreeNode root) {

        if (root == null) return 0;

        int ans = 0;
        Queue<Pair> q = new LinkedList<>();
        q.offer(new Pair(root,0));
        while (!q.isEmpty()){
            int size = q.size();
            int mmin = q.peek().num; // to make the id starting from Zero
            int first = 0;
            int last = 0;
            for (int i =0; i < size; i++){
                int cur_id = q.peek().num - mmin;
                TreeNode node = q.peek().node;
                q.poll();
                if (i == 0){
                    first = cur_id;
                }
                if (i == size-1){
                    last = cur_id;
                }
                if (node.left != null){
                    q.offer(new Pair(node.left, cur_id*2+1));
                }
```

```
                if (node.right != null){
                    q.offer(new Pair(node.right, cur_id*2+2));
                }
            }
            ans = Math.max(ans, last-first+1);
        }
        return ans;
    }
}
```

Check for Children Sum Property in a Binary Tree

https://www.geeksforgeeks.org/problems/children-sum-parent/1

```
//User function Template for Java


/*Complete the function below
Node is as follows:
class Node{
        int data;
        Node left,right;

        Node(int key)
        {
            data = key;
            left = right = null;
        }
}

*/
class Solution
{
    //Function to check whether all nodes of a tree have the value
    //equal to the sum of their child nodes.
    public static int isSumProperty(Node root)
    {
        // add your code here
        if (root == null) return 0;
        return isSumPropertyUtil(root);
    }

    private static int isSumPropertyUtil(Node root){
```

```
        int leftValue = 0;
        int rightValue = 0;

        if (root == null
            || (root.left == null && root.right == null))
            return 1;

        else{
            if (root.left != null){
                leftValue = root.left.data;
            }
            if (root.right != null){
                rightValue = root.right.data;
            }

            if (root.data == leftValue + rightValue
                    && isSumPropertyUtil(root.left) != 0
                    && isSumPropertyUtil(root.right) != 0)

                return 1;

            else return 0;
        }
    }
}
```

All Nodes Distance K in Binary Tree

https://leetcode.com/problems/all-nodes-distance-k-in-binary-tree/

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
class Solution {
    Map<TreeNode, TreeNode> parent_track;
    public List<Integer> distanceK(TreeNode root, TreeNode target, int k) {
```

```
        parent_track = new HashMap<>();
        markParents(root,target);
        Map<TreeNode, Boolean> visited = new HashMap<>();
        Queue<TreeNode> q = new LinkedList<>();
        q.offer(target);
        visited.put(target, true);
        int curr_level = 0;

        while (!q.isEmpty()){
            int size = q.size();
            if (curr_level == k) break;
            curr_level++;

            for (int i =0; i < size; i++){
                TreeNode curr = q.poll();
                if (curr.left != null && visited.get(curr.left) == null){
                    q.offer(curr.left);
                    visited.put(curr.left,true);
                }

                if (curr.right != null && visited.get(curr.right) == null){
                    q.offer(curr.right);
                    visited.put(curr.right, true);
                }

                if (parent_track.get(curr) != null &&
visited.get(parent_track.get(curr)) == null){
                    q.offer(parent_track.get(curr));
                    visited.put(parent_track.get(curr),true);
                }
            }
        }

        List<Integer> res = new ArrayList<>();
        while (!q.isEmpty()){
            TreeNode curr = q.poll();
            res.add(curr.val);
        }
        return res;
    }

    private void markParents(TreeNode root, TreeNode target){
        Queue<TreeNode> q = new LinkedList<>();
        q.offer(root);
        while (!q.isEmpty()){
            TreeNode curr = q.poll();
```

```
            if (curr.left != null){
                parent_track.put(curr.left, curr);
                q.offer(curr.left);
            }
            if (curr.right != null){
                parent_track.put(curr.right, curr);
                q.offer(curr.right);
            }
        }
    }
}
```

Amount of Time for Binary Tree to be Infected

https://leetcode.com/problems/amount-of-time-for-binary-tree-to-be-infected/description/

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */

class Solution {
    Map<Integer,List<Integer>> graph;
    int time;
    public int amountOfTime(TreeNode root, int start) {
        graph  = new HashMap<>();
        time = 0;
        makeAdjacentGraph(root);
        BFS(start);
        return time-1;
    }

    private void makeAdjacentGraph(TreeNode root){
```

```
        if (root == null)  return;

        List<Integer> neighboursRoot = graph.getOrDefault(root.val, new
ArrayList<>());

        TreeNode left = root.left;
        TreeNode right = root.right;

        if (left != null){
            neighboursRoot.add(left.val);
            List<Integer> nbrsLeft = graph.getOrDefault(left.val, new
ArrayList<>());
            nbrsLeft.add(root.val);
            graph.put(left.val,nbrsLeft);
        }

        if (right != null){
            neighboursRoot.add(right.val);
            List<Integer> nbrsRight = graph.getOrDefault(right.val, new
ArrayList<>());
            nbrsRight.add(root.val);
            graph.put(right.val,nbrsRight);
        }

        graph.put(root.val, neighboursRoot);
        makeAdjacentGraph(root.left);
        makeAdjacentGraph(root.right);
    }

    private void BFS(int root){
        Queue<Integer> q = new LinkedList<>();
        q.add(root);
        Set<Integer> visited = new HashSet<>();

        while (!q.isEmpty()){
            int size = q.size();
            while (size > 0){
                int curr = q.poll();
                visited.add(curr);
                List<Integer> nbrs = graph.get(curr);
                for (int next:nbrs){
                    if (!visited.contains(next)){
                        q.add(next);
                    }
                }
                size--;
```

```
            }
            time++;
        }
    }


}
```

Count Total Nodes in a Complete Binary Tree

```
Formula: 2^h-1
```

https://leetcode.com/problems/count-complete-tree-nodes/description/

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public int countNodes(TreeNode root) {

        return countNodesUtil(root);
    }

    private int countNodesUtil(TreeNode root){

        if (root == null) return 0;

        int leftHeight = getLeftHeight(root);
        int rightHeight = getRightHeight(root);

        // if left and right are equal it means that the tree is complete
```

```
binary tree
        if (leftHeight == rightHeight){
            return ((2<<(leftHeight)) - 1);
        }

        // else recursively calculate the number of nodes in left and right
        else{
            return countNodesUtil(root.left) + countNodesUtil(root.right) + 1;
        }
    }

    private int getLeftHeight(TreeNode root){
        int count = 0;
        while (root.left != null){
            root = root.left;
            count++;
        }
        return count;
    }

    private int getRightHeight(TreeNode root){
        int count = 0;
        while (root.right != null){
            root = root.right;
            count++;
        }
        return count;
    }
}
```

Requirements Needed to construct a Unique Binary Tree

https://leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/description/

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
```

```
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    Map<Integer, Integer> map;
    public TreeNode buildTree(int[] preorder, int[] inorder) {
        map = new HashMap<>();
        int i = 0;
        for (int ele : inorder){
            map.put(ele,i++);
        }
        return buildTreeUtil(preorder,0, preorder.length-1, inorder, 0,
inorder.length-1);
    }

    private TreeNode buildTreeUtil(int []preOrder, int preStart, int preEnd,
int []inOrder, int inStart, int inEnd){

        if (preStart > preEnd || inStart > inEnd){
            return null;
        }

        TreeNode root = new TreeNode(preOrder[preStart]);

        int inRoot = map.get(root.val);
        int numsLeft = inRoot - inStart;

        root.left = buildTreeUtil(preOrder, preStart + 1, preStart + numsLeft,
                                            inOrder, inStart, inRoot -1);

        root.right = buildTreeUtil(preOrder, preStart + numsLeft + 1, preEnd,
                                            inOrder, inRoot + 1, inEnd);

        return root;
    }
}
```

Construct a Binary Tree from Post Order and In Order Traversal

https://leetcode.com/problems/construct-binary-tree-from-inorder-and-postorder-traversal/description/

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    Map<Integer,Integer> map;
    public TreeNode buildTree(int[] inorder, int[] postorder) {

        if (inorder == null || postorder == null || inorder.length != 
postorder.length){
            return null;
        }
        map = new HashMap<>();

        int i = 0;
        for (int ele : inorder){
            map.put(ele,i);
            i++;
        }

        return buildTreeUtil(inorder, 0, inorder.length-1, postorder, 0, 
postorder.length-1);
    }

    private TreeNode buildTreeUtil(int []inorder, int inStart, int inEnd, int 
[]postorder, int poStart, int poEnd){

        if (poStart > poEnd || inStart > inEnd){
            return null;
        }

        TreeNode root = new TreeNode(postorder[poEnd]);

        int inRoot = map.get(postorder[poEnd]);
        int numsLeft = inRoot - inStart;
```

```
        root.left = buildTreeUtil(inorder, inStart, inRoot -1,
                                  postorder, poStart, poStart + numsLeft-1);
        root.right = buildTreeUtil(inorder, inRoot +1, inEnd,
                                  postorder, poStart + numsLeft,  poEnd -1);

        return root;
    }
}
```

Serialize and De-serialize Binary Tree

https://leetcode.com/problems/serialize-and-deserialize-binary-tree/description/

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Codec {

    // Encodes a tree to a single string.
    public String serialize(TreeNode root) {

        if (root == null){
            return "";
        }

        Queue<TreeNode> q = new LinkedList<>();
        StringBuilder res = new StringBuilder();
        q.add(root);

        while (!q.isEmpty()){
            TreeNode node = q.poll();
            if (node == null){
                res.append("n ");
                continue;
            }
            res.append(node.val+ " ");
            q.add(node.left);
```

```
            q.add(node.right);
        }

        return res.toString();
    }

    // Decodes your encoded data to tree.
    public TreeNode deserialize(String data) {
        if (data == "") return null;
        Queue<TreeNode> q = new LinkedList<>();
        String[] values = data.split(" ");
        TreeNode root = new TreeNode(Integer.parseInt(values[0]));
        q.add(root);

        for (int i = 1; i < values.length; i++){
            TreeNode parent = q.poll();
            if (!values[i].equals("n")){
                TreeNode left = new TreeNode(Integer.parseInt(values[i]));
                parent.left = left;
                q.add(left);
            }
            if (!values[++i].equals("n")){
                TreeNode right = new TreeNode(Integer.parseInt(values[i]));
                parent.right = right;
                q.add(right);
            }
        }

        return root;
    }
}

// Your Codec object will be instantiated and called as such:
// Codec ser = new Codec();
// Codec deser = new Codec();
// TreeNode ans = deser.deserialize(ser.serialize(root));
```

Morris Traversal - Inorder | Preorder

```
private void preOrderMorrisTraversalUtil(TreeNode root){

        TreeNode curr = root;

        while( curr != null){
```

```
            if (curr.left == null){
                res.add(curr.val);
                curr = curr.right;
            }
            else{
                TreeNode prev = curr.left;
                while (prev.right != null && prev.right != curr){
                    prev = prev.right;
                }

                if (prev.right == null){
                    res.add(curr.val);
                    prev.right = curr;
                    curr = curr.left;
                }
                else{
                    prev.right = null;
                    curr = curr.right;
                }
            }
        }
    }
```

```
private void inorderMorrisTraversalUtil(TreeNode root){

        TreeNode curr = root;

        while( curr != null){
            if (curr.left == null){
                res.add(curr.val);
                curr = curr.right;
            }
            else{
                TreeNode prev = curr.left;
                while (prev.right != null && prev.right != curr){
                    prev = prev.right;
                }

                if (prev.right == null){
                    prev.right = curr;
                    curr = curr.left;
                }
                else{
                    prev.right = null;
                    res.add(curr.val);
```

```
                    curr = curr.right;
                }
            }
        }
    }
```

Flatten a Binary Tree to Linked List

```
Right, Left, Root
```

https://leetcode.com/problems/flatten-binary-tree-to-linked-list/description/

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    TreeNode prev;
    public void flatten(TreeNode root) {
        prev = null;
        flattenUtil(root);
    }

    private void flattenUtil(TreeNode root){

        // base case
        if (root == null) return;

        flattenUtil(root.right);
        flattenUtil(root.left);

        root.right = prev;
```

```
            root.left = null;
            prev = root;

        }
    }
```

Binary Search Tree

Search in a BST

https://leetcode.com/problems/search-in-a-binary-search-tree/

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public TreeNode searchBST(TreeNode root, int val) {

        return searchBSTUtil(root,val);
    }

    private TreeNode searchBSTUtil(TreeNode root, int val){

        while (root != null && root.val != val){
            if (val < root.val){
                root = root.left;
            }
            else{
```

```
                root = root.right;
            }
        }

        return root;
    }
}
```

Ciel in a Binary Search Tree

https://www.geeksforgeeks.org/problems/implementing-ceil-in-bst/1

```
// User function Template for Java

class Tree {
    // Function to return the ceil of given number in BST.
    int findCeil(Node root, int key) {
        if (root == null) return -1;
        // Code here
        return findCeilUtil(root,key);
    }

    int findCeilUtil(Node root, int val){
        int ceil = -1;
        while (root != null){
            if (root.data == val){
                ceil = root.data;
                return ceil;
            }

            else if (val > root.data){
                root = root.right;
            }
            else{
                ceil = root.data;
                root = root.left;
            }
        }

        return ceil;
    }
}
```

Floor in a BST

https://www.geeksforgeeks.org/problems/floor-in-bst/1

```java
// User function Template for Java

class Solution {
    public static int floor(Node root, int x) {
        // Code here
        return findFloorUtil(root,x);
    }

    static int findFloorUtil(Node root, int val){
        int floor = -1;
        while (root != null){
            if (root.data == val){
                floor = root.data;
                return floor;
            }

            else if (val > root.data){
                floor = root.data;
                root = root.right;
            }
            else{
                root = root.left;
            }
        }

        return floor;
    }
}
```

Insert a given node in BST

https://leetcode.com/problems/insert-into-a-binary-search-tree/description/

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
```

```
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public TreeNode insertIntoBST(TreeNode root, int val) {

        return insertIntoBSTUtil(root, val);
    }

    private TreeNode insertIntoBSTUtil(TreeNode root, int val){

        // base case
        if (root == null){
            return new TreeNode(val);
        }

        // keep the track of root
        TreeNode curr = root;

        while (true){
            if (curr.val <= val){
                if (curr.right != null){
                    curr = curr.right;
                }
                else{
                    curr.right = new TreeNode(val);
                    break;
                }
            }
            else{
                if (curr.left != null){
                    curr = curr.left;
                }
                else{
                    curr.left = new TreeNode(val);
                    break;
                }
            }
        }
        return root;
```

```
    }
}
```

Delete a node in BST

https://leetcode.com/problems/delete-node-in-a-bst/description/

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public TreeNode deleteNode(TreeNode root, int key) {

        return deleteNodeUtil(root, key);
    }

    private TreeNode deleteNodeUtil(TreeNode root, int key){

        // base case
        if (root == null){
            return null;
        }

        if (root.val == key){
            return helperDeleteUtil(root);
        }

        TreeNode dummy = root;
        while (root != null){
            // if key is smaller than root
            if (root.val > key){
                if (root.left != null && root.left.val == key){
```

```
                    root.left = helperDeleteUtil(root.left);
                    break;
                }
                else{
                    root = root.left;
                }
            }
            // key is greater than root
            else{
                if (root.right != null && root.right.val == key){
                    root.right = helperDeleteUtil(root.right);
                    break;
                }
                else{
                    root = root.right;
                }
            }
        }
        return dummy;
    }

    private TreeNode helperDeleteUtil(TreeNode root){

        if (root.left == null){
            return root.right;
        }

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

        TreeNode rightChild = root.right;
        TreeNode lastRight = findLastRightUtil(root.left); // this method will
return the last right child from the left subtree
        lastRight.right = rightChild;
        return root.left;
    }

    private TreeNode findLastRightUtil(TreeNode root){
        if (root.right == null){
            return root;
        }

        return findLastRightUtil(root.right);
    }
}
```

Kth Smallest Element in BST

https://leetcode.com/problems/kth-smallest-element-in-a-bst/description/

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    int cnt = 0;
    int res = -1;
    public int kthSmallest(TreeNode root, int k) {

        recursiveInorderTraversalUtil(root,k);
        return res;
    }

    private void recursiveInorderTraversalUtil(TreeNode root, int k){

        // base case
        if (root == null){
            return;
        }
        recursiveInorderTraversalUtil(root.left, k);
        cnt++;
        if (cnt == k){
            res = root.val;
            return;
        }

        recursiveInorderTraversalUtil(root.right, k);
    }

}
```

Check if a Tree is BST | Validate a BST

https://leetcode.com/problems/validate-binary-search-tree/

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public boolean isValidBST(TreeNode root) {

        return isValidBST(root, Long.MIN_VALUE, Long.MAX_VALUE);
    }

    private boolean isValidBST(TreeNode root, long minVal, long maxVal){

        if (root == null) return true;
        if (root.val >= maxVal || root.val <= minVal) return false;

        return isValidBST(root.left, minVal, root.val) &&
isValidBST(root.right, root.val, maxVal);
    }
}
```

Find Longest Common Ancestor in BST

https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-search-tree/description/

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
```

```
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */

class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode
q) {

        return lowestCommonAncestorUtil(root, p, q);
    }

    private TreeNode lowestCommonAncestorUtil (TreeNode root, TreeNode p,
TreeNode q){

        if (root == null) return null;

        int curr = root.val;
        if (curr < p.val && curr < q.val){
            return lowestCommonAncestorUtil(root.right, p, q);
        }
        if (curr > p.val && curr > q.val){
            return lowestCommonAncestorUtil(root.left, p, q);
        }
        return root;
    }
}
```

Construct a BST from Preorder Traversal

https://leetcode.com/problems/construct-binary-search-tree-from-preorder-traversal/description/

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
```

```
 *     }
 * }
 */
class Solution {
    int nodeIndex;
    public TreeNode bstFromPreorder(int[] preorder) {

        nodeIndex = 0;
        int start = Integer.MIN_VALUE;
        int end = Integer.MAX_VALUE;
        return bstFromPreorderUtil(preorder, nodeIndex, end);
    }

    private TreeNode bstFromPreorderUtil(int []preOrder, int start, int end){

        if (nodeIndex == preOrder.length || preOrder[nodeIndex] < start ||
preOrder[nodeIndex] > end) {
            return null;
        }

        int val = preOrder[nodeIndex++];
        TreeNode root = new TreeNode(val);
        root.left = bstFromPreorderUtil(preOrder, start, val);
        root.right = bstFromPreorderUtil(preOrder, val, end);
        return root;
    }
}
```

Inorder Successor in BST

https://www.geeksforgeeks.org/problems/inorder-successor-in-bst/1

```
//User function Template for Java

/*Complete the function below
Node is as follows:
class Node{
        int data;
        Node left,right;
        Node(int d){
                data=d;
                left=right=null;
        }
}
```

```
*/
class Solution
{
    // returns the inorder successor of the Node x in BST (rooted at 'root')
        public Node inorderSuccessor(Node root,Node x)
         {
          //add code here.
          return inorderSuccessorUtil(root, x);
         }

        private Node inorderSuccessorUtil(Node root, Node x){

            Node successor = null;

            while (root != null){
                if (x.data >= root.data){
                    root = root.right;
                }
                else{
                    successor = root;
                    root = root.left;
                }
            }

            return successor;
        }
}
```

BST Iterator

https://leetcode.com/problems/binary-search-tree-iterator/description/

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
```

```
 *     }
 * }
 */
class BSTIterator {

    private Stack<TreeNode> st = new Stack<>();

    public BSTIterator(TreeNode root) {
        pushAll(root);
    }

    public int next() {

        TreeNode tempNode = st.pop();
        pushAll(tempNode.right);
        return tempNode.val;
    }

    public boolean hasNext() {

        return (!st.isEmpty());
    }

    private void pushAll(TreeNode root){

        while (root != null){
            st.push(root);
            root = root.left;
        }
    }
}
```

Two Sum in BST

https://leetcode.com/problems/two-sum-iv-input-is-a-bst/description/

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
```

```
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class BSTIterator {

    private Stack<TreeNode> st = new Stack<>();
    boolean reverse = true;

    public BSTIterator(TreeNode root, boolean isReverse) {
        reverse = isReverse;
        pushAll(root);
    }

    public int next() {

        TreeNode tempNode = st.pop();
        if (reverse == false){
            pushAll(tempNode.right);
        }
        else pushAll(tempNode.left);
        return tempNode.val;
    }

    public boolean hasNext() {

        return (!st.isEmpty());
    }

    private void pushAll(TreeNode root){

        while (root != null){
            st.push(root);
            if (reverse == true){
                root = root.right;
            }
            else {
                root = root.left;
            }
        }
    }
}
```

```java
class Solution {
    public boolean findTarget(TreeNode root, int k) {
        BSTIterator left = new BSTIterator(root, false);
        BSTIterator right = new BSTIterator(root, true);

        int i = left.next();
        int j = right.next();

        while (i < j){
            if (i + j == k){
                return true;
            }
            else if (i + j < k){
                i = left.next();
            }
            else{
                j = right.next();
            }
        }
        return false;
    }
}
```

Recover BST

https://leetcode.com/problems/recover-binary-search-tree/description/

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
```

```java
    private TreeNode first;
    private TreeNode prev;
    private TreeNode middle;
    private TreeNode last;
    public void recoverTree(TreeNode root) {

        first = middle = last = null;
        prev = new TreeNode(Integer.MIN_VALUE);
        inorder(root);

        // if find two violations
        if (first != null && last != null){
            // swap 'first' and 'last'
            swapUtil(first,last);
        }
        // if find one violation
        else if (first != null && middle != null){
            swapUtil(first, middle);
        }
    }

    private void inorder(TreeNode root){
        if (root == null){
            return;
        }
        inorder(root.left);

        if (prev != null && (root.val < prev.val)){
            // if this is first violation, mark these two nodes as
            // 'first' and 'middle'
            if (first == null){
                first = prev;
                middle = root;
            }
            // If this is second violation, mark this node as last
            else{
                last = root;
            }
        }
         // Mark this node as previous
            prev = root;
            inorder(root.right);
    }

    private void swapUtil(TreeNode node1, TreeNode node2){
        int temp = node1.val;
```

```
        node1.val = node2.val;
        node2.val = temp;
    }
}
```

Largest BST in Binary Tree

https://www.geeksforgeeks.org/problems/largest-bst/1

```
class NodeValue{
    public int maxNode, minNode, maxSize;

    NodeValue(int minNode, int maxNode, int maxSize){
        this.maxNode = maxNode;
        this.minNode = minNode;
        this.maxSize = maxSize;
    }
};

class Solution{

    // Return the size of the largest sub-tree which is also a BST
    static int largestBst(Node root)
    {
        // Write your code here
        return largestBstHelper(root).maxSize;

    }

    private static NodeValue largestBstHelper(Node root){

        // An empty tree is a BST of size 0
        if (root == null){
            return new NodeValue(Integer.MAX_VALUE, Integer.MIN_VALUE, 0);
        }

        // Get values from left and right subtree of current tree
        NodeValue left = largestBstHelper(root.left);
        NodeValue right = largestBstHelper(root.right);

        // Current node is greater than max in left AND smaller than min in
right,
        if (left.maxNode < root.data && root.data < right.minNode){
```

```
            // It is a BST
            int min = Math.min(root.data, left.minNode);
            int max = Math.max(root.data, right.maxNode);
            int size = left.maxSize + right.maxSize + 1;

            return new NodeValue(min, max, size);
        }

        // Otherwise, return [-inf, inf] so that parent can't be valid BST
        int min = Integer.MIN_VALUE;
        int max = Integer.MAX_VALUE;
        int size = Math.max(left.maxSize, right.maxSize);

        return new NodeValue(min, max, size);
    }

}
```