Cameras In Binary Tree | Leetcode 968 Binary Tree Cameras

```
public static int MinCamerasInBT(TreeNode root) {

    if(MinCamerasInBT_(root) == -1) cameras++;
    return cameras;
}
```

```
public static int cameras = 0;

public static int MinCamerasInBT_(TreeNode root) {
    if(root == null) return 1;

    int lchild = MinCamerasInBT_(root.left);
    int rchild = MinCamerasInBT_(root.right);

    if(lchild == -1 || rchild == -1){
        cameras++;
        return 0;
    }

    if(lchild == 0 || rchild == 0) return 1;

    return -1;
}

public static int MinCamerasInBT(TreeNode root) {

    if(MinCamerasInBT_(root) == -1) cameras++;
    return cameras;
}
```

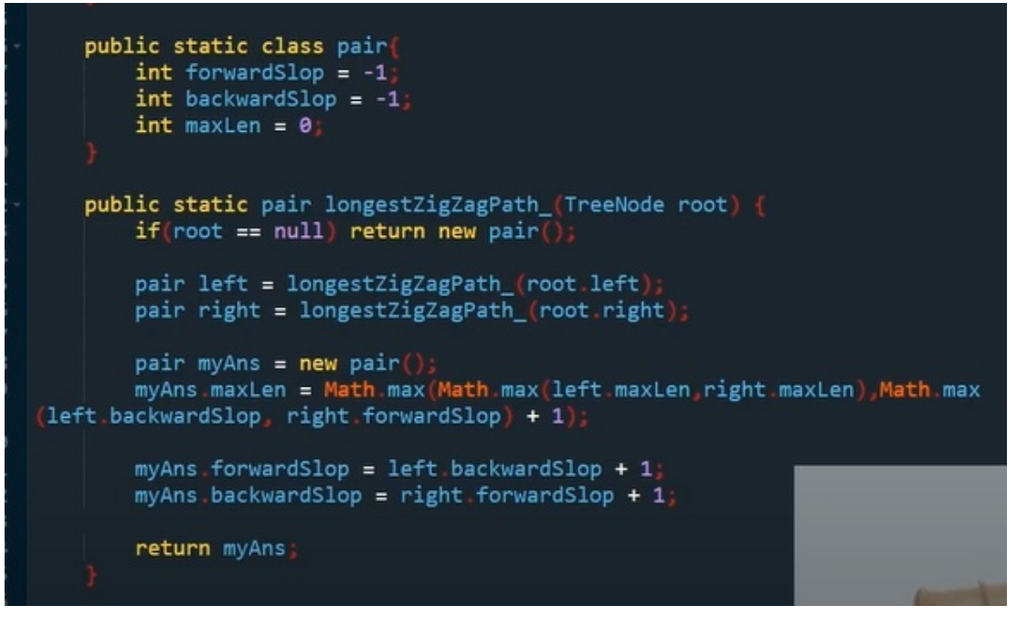
House Robber In Binary Tree | Leetcode 337 House Robber III

```java
public static class housePair{
    int withRobbery = 0;
    int withoutRobbery = 0;
}

public static housePair HouseRobber_(TreeNode root) {
    if(root==null) return new housePair();

    housePair left = HouseRobber_(root.left);
    housePair right = HouseRobber_(root.right);

    housePair myAns = new housePair();

    myAns.withRobbery = left.withoutRobbery + root.val + right
.withoutRobbery;

    myAns.withoutRobbery = Math.max(left.withRobbery,left.withoutRobbery)
+ Math.max(right.withRobbery,right.withoutRobbery);

    return myAns;
}



public static int HouseRobber(TreeNode root) {
    housePair res = HouseRobber_(root);

    return Math.max(res.withRobbery,res.withoutRobbery);
}
```

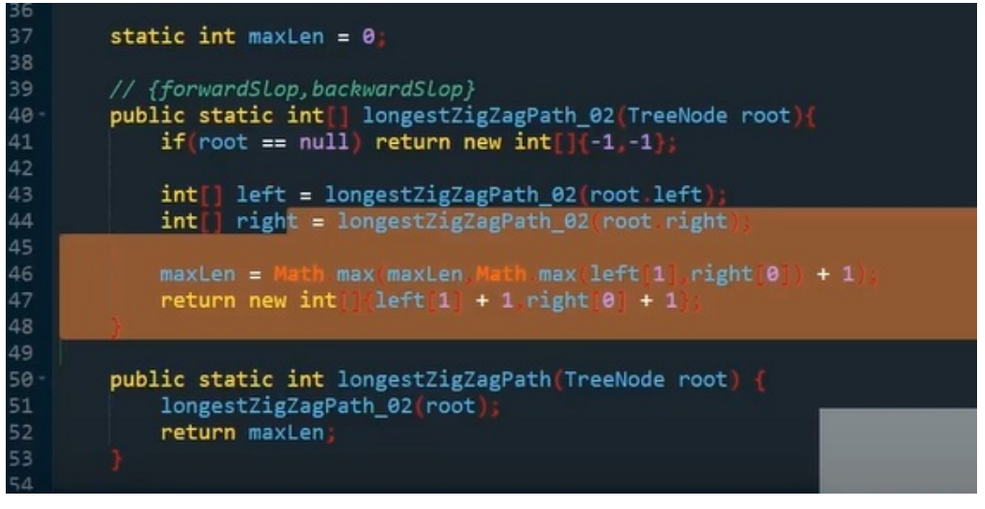
other way

```java
// {with-Robbery Maximum Amount, without-Robbery maximum Amount}
public static int[] HouseRobber_(TreeNode root) {
    if(root==null) return new int[2];

    int[] left = HouseRobber_(root.left);
    int[] right = HouseRobber_(root.right);

    int[] myAns= new int[2];

    myAns[0] = left[1] + root.val +right[1];
    myAns[1] = Math.max(left[0],left[1]) + Math.max(right[0],right[1
    return myAns;
}
```

Longest Zig Zag Path In Binary Tree | Leetcode 1372. Longest ZigZag Path in a Binary Tree

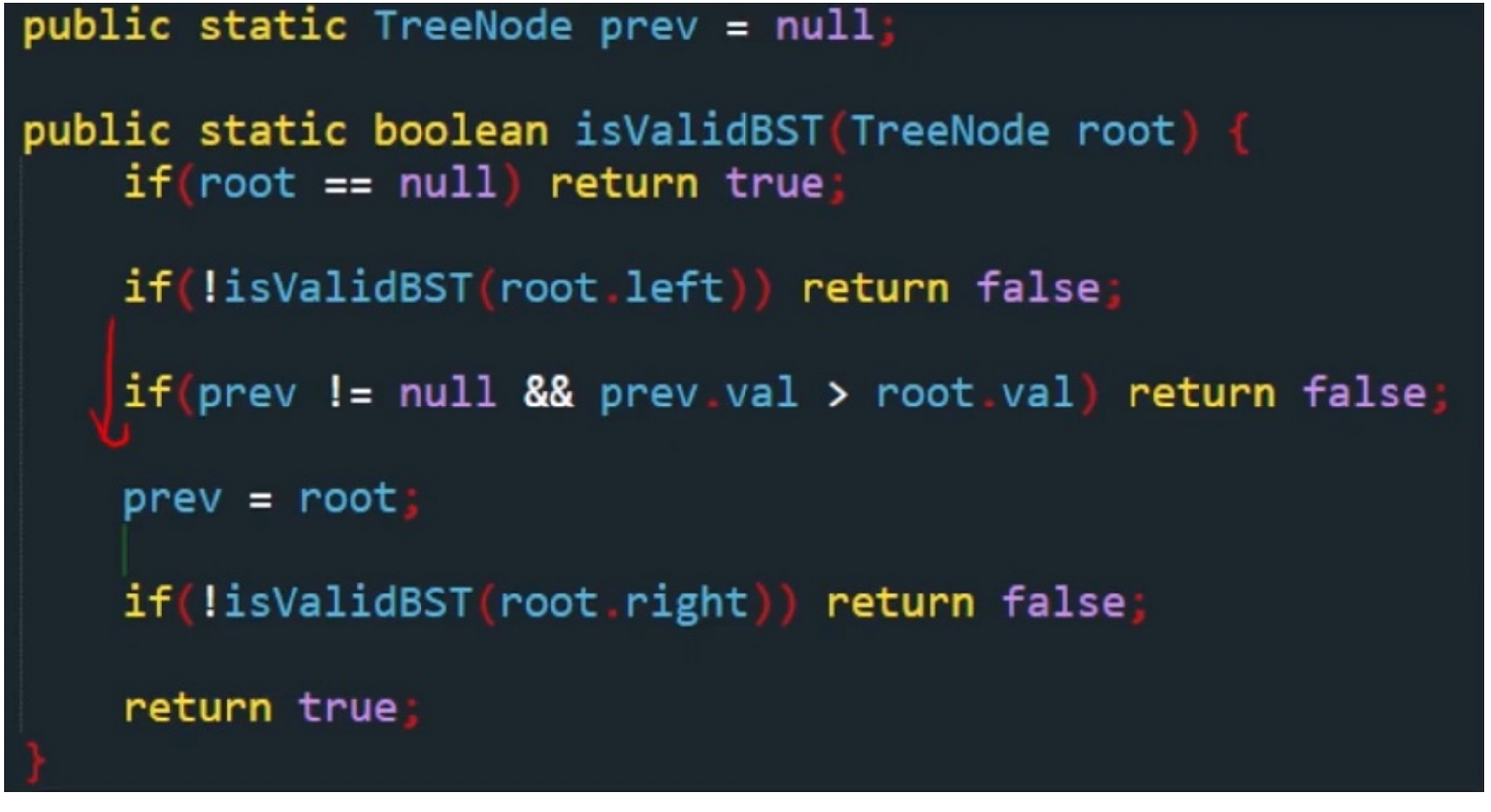


```
    public static class pair{
        int forwardSlop = -1;
        int backwardSlop = -1;
        int maxLen = 0;
    }

    public static pair longestZigZagPath_(TreeNode root) {
        if(root == null) return new pair();

        pair left = longestZigZagPath_(root.left);
        pair right = longestZigZagPath_(root.right);

        pair myAns = new pair();
        myAns.maxLen = Math.max(Math.max(left.maxLen,right.maxLen),Math.max
(left.backwardSlop, right.forwardSlop) + 1);

        myAns.forwardSlop = left.backwardSlop + 1;
        myAns.backwardSlop = right.forwardSlop + 1;

        return myAns;
    }
```



```
36
37      static int maxLen = 0;
38
39      // {forwardSlop,backwardSlop}
40      public static int[] longestZigZagPath_02(TreeNode root){
41          if(root == null) return new int[]{-1,-1};
42
43          int[] left = longestZigZagPath_02(root.left);
44          int[] right = longestZigZagPath_02(root.right);
45
46          maxLen = Math.max(maxLen,Math.max(left[1],right[0]) + 1);
47          return new int[]{left[1] + 1,right[0] + 1};
48      }
49
50      public static int longestZigZagPath(TreeNode root) {
51          longestZigZagPath_02(root);
52          return maxLen;
53      }
54
```

Validate Bst



recover bst

```java
public static class TreeNode {
    int val = 0;
    TreeNode left = null;
    TreeNode right = null;

    TreeNode(int val) {
        this.val = val;
    }
}

static TreeNode a = null, b = null, prev = null;

public static boolean recoverTree_(TreeNode root) {

    if (root == null)
        return true;

    if (!recoverTree_(root.left))
        return false;

    if (prev != null && prev.val > root.val) {
        b = root;
        if (a == null)
            a = prev;
        else
            return false;
    }

    prev = root;
    if (!recoverTree_(root.right))
        return false;

    return true;

}

public static void recoverTree(TreeNode root) {
    recoverTree_(root);
    if (a != null) {
        int temp = a.val;
        a.val = b.val;
        b.val = temp;
    }
}
```

Serailize and Deserialize of Binary Tree Part 1 | Leetcode 2

```java
public static void serialize(TreeNode root, StringBuilder sb){
    if(root == null){
        sb.append("null,");
        return;
    }

    sb.append(root.val+",");
    serialize(root.left, sb);
    serialize(root.right, sb);
}

// Encodes a tree to a single string.
public static String serialize(TreeNode root) {
    StringBuilder sb = new StringBuilder();
    serialize(root,sb);
    return sb.toString();
}
```

```java
34      static int idx = 0;
35      public static TreeNode deserialize(String[] arr) {
36          if(idx >= arr.length || arr[idx].equals("null")){
37              idx++;
38              return null;
39          }
40
41          TreeNode node = new TreeNode(arr[idx++]);
42          node.left = deserialize(arr);
43          node.right = deserialize(arr);
44
45          return node;
46      }
47
48      // Decodes your encoded data to tree.
49      public static TreeNode deserialize(String str) {
50          String[] arr = str.split(",");
51          deserialize(arr);
52      }
```

left view

```java
public static ArrayList<Integer> leftView(TreeNode root) {
    if(root == null) return null;
    ArrayList<Integer> ans = new ArrayList<>();
    LinkedList<TreeNode> que = new LinkedList<>();
    que.addLast(root);
    while(que.size() != 0){
        int size = que.size();
        ans.add(que.getFirst().val);
        while(size-->0){
            TreeNode rn = que.removeFirst(); // rn : remove node
            if(rn.left != null) que.addLast(rn.left);
            if(rn.right != null) que.addLast(rn.right);
        }
    }

    return ans;
}
```

right view

```java
public static ArrayList<Integer> rightView(TreeNode root) {
    LinkedList<TreeNode> que = new LinkedList<>();
    que.addLast(root);
    ArrayList<Integer> ans = new ArrayList<>();

    while(que.size() != 0){
        int size = que.size();
        ans.add(que.getFirst().val);
        while(size-- > 0){
            TreeNode rn = que.removeFirst();

            if(rn.right != null) que.addLast(rn.right);
            if(rn.left != null) que.addLast(rn.left);
        }
    }


    return ans;
}
```

Width Of Shadow Of Binary Tree

```
// {minHt, maxHt} where Ht is horizontal level.
public static void width(TreeNode root,int hl,int[] ans) {
    if(root == null) return;

    ans[0] = Math.min(ans[0],hl);
    ans[0] = Math.max(ans[1],hl);

    width(root.left,hl - 1,ans);
    width(root.right,hl + 1,ans);
}

public static int width(TreeNode root) {
   int[] ans = new int[2];
   width(root,0,ans);
   return ans[1] - ans[0] + 1;
}
```

```java
public static class verticalPair {
    TreeNode node = null;
    int hl = 0; // horizontal Level

    verticalPair(TreeNode node, int hl) {
        this.node = node;
        this.hl = hl;
    }
}

public static ArrayList<ArrayList<Integer>> verticalOrderTraversal(TreeNode root) {
    LinkedList<verticalPair> que = new LinkedList<>();
    que.addLast(new verticalPair(root, 0));
    HashMap<Integer, ArrayList<Integer>> map = new HashMap<>();

    int minHL = 0;
    int maxHL = 0;

    while (que.size() != 0) {
        int size = que.size();
        while (size-- > 0) {
            verticalPair rp = que.removeFirst();

            map.putIfAbsent(rp.hl, new ArrayList<>());
            // if (!map.containsKey(rp.hl))
            // map.put(rp.hl, new ArrayList<>());

            map.get(rp.hl).add(rp.node.val);

            minHL = Math.min(minHL, rp.hl);
            maxHL = Math.max(maxHL, rp.hl);

            if (rp.node.left != null)
                que.addLast(new verticalPair(rp.node.left, rp.hl - 1));

            if (rp.node.right != null)
                que.addLast(new verticalPair(rp.node.right, rp.hl + 1));
        }
    }

    ArrayList<ArrayList<Integer>> ans = new ArrayList<>();
    while (minHL <= maxHL) {
        ans.add(map.get(minHL));
        minHL++;
    }

    return ans;
}
```

```java
public static class verticalPair {
    TreeNode node = null;
    int hl = 0; // horizontal Level

    verticalPair(TreeNode node, int hl) {
        this.node = node;
        this.hl = hl;
    }
}

// ans = {minHL,maxHL}
public static void width(TreeNode root, int hl, int[] ans) {
    if (root == null)
        return;

    ans[0] = Math.min(hl, ans[0]);
    ans[1] = Math.max(hl, ans[1]);

    width(root.left, hl - 1, ans);
    width(root.right, hl + 1, ans);
}

public static ArrayList<ArrayList<Integer>> verticalOrderTraversal(TreeNode r
    PriorityQueue<verticalPair> que = new PriorityQueue<>((a, b) -> {
        return a.node.val - b.node.val; // this - other for default behaviour
    });
    PriorityQueue<verticalPair> childQue = new PriorityQueue<>((a, b) -> {
        return a.node.val - b.node.val;
    });

    int[] minMax = new int[2];
    width(root, 0, minMax);
    int length = minMax[1] - minMax[0] + 1;
    ArrayList<ArrayList<Integer>> ans = new ArrayList<>();
    for (int i = 0; i < length; i++)
        ans.add(new ArrayList<>());

    que.add(new verticalPair(root, -minMax[0]));

    while (que.size() != 0) {
        verticalPair rp = que.remove();

        ans.get(rp.hl).add(rp.node.val);

        if (rp.node.left != null)
            childQue.add(new verticalPair(rp.node.left, rp.hl - 1));

        if (rp.node.right != null)
            childQue.add(new verticalPair(rp.node.right, rp.hl + 1));

        if (que.size() == 0) {
            PriorityQueue<verticalPair> temp = que;
            que = childQue;
            childQue = temp;
        }
    }

    return ans;

}
```

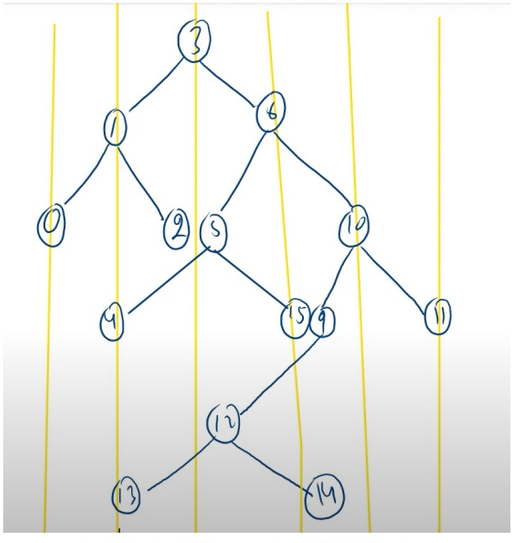
```java
You, 21 hours ago | 1 author (You)
class Pair implements Comparable<Pair> {
    TreeNode node;
    int key;
    int level = 0;

    public Pair() {

    }

    public Pair(TreeNode node, int key, int level) {
        this.node = node;
        this.key = key;
        this.level = level;

    }

    // make this note
    @Override
    public int compareTo(Pair obj) {
        if (this.level == obj.level)
            return this.node.val - obj.node.val;

        return this.level - obj.level;
    }
}
```

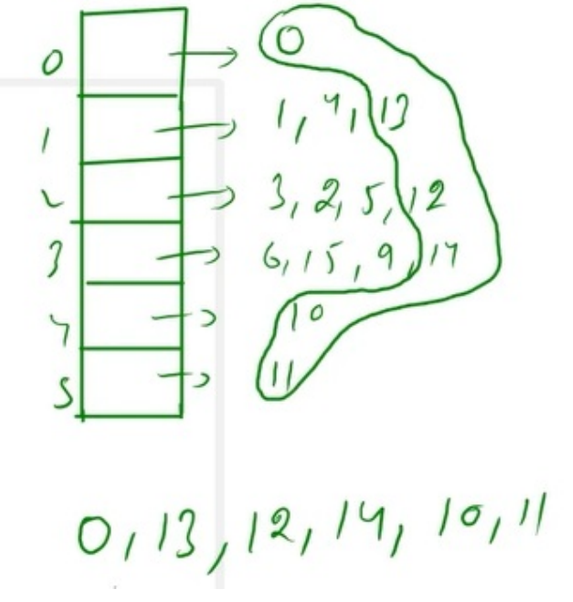
```java
int min = 0;
int max = 0;

public void getWidth(TreeNode root, int level) {
    if (root == null) {
        return;
    }
    min = Math.min(min, level);
    max = Math.max(max, level);
    getWidth(root.left, level - 1);
    getWidth(root.right, level + 1);

}
```

```java
public List<List<Integer>> verticalTraversal(TreeNode root) {

    // getwidth
    getWidth(root, level: 0);
    int width = max - min + 1;
    HashMap<Integer, ArrayList<Integer>> map = new HashMap<>();

    for (int i = 0; i < width; i++) {
        map.put(i, new ArrayList<>());
    }
    PriorityQueue<Pair> q = new PriorityQueue<>();
    q.add(new Pair(root, Math.abs(min), level: 0));
    while (q.size() > 0) {      You, 21 hours ago • Updated code on timestamp:  12-
        int size = q.size();
        while (size-- > 0) {
            Pair rem = q.remove();
            map.get(rem.key).add(rem.node.val);
            if (rem.node.left != null) {
                q.add(new Pair(rem.node.left, rem.key - 1, rem.level + 1));
            }
            if (rem.node.right != null) {
                q.add(new Pair(rem.node.right, rem.key + 1, rem.level + 1));
            }
        }
    }
    // make ans
    List<List<Integer>> ans = new ArrayList<>();

    for (int i = 0; i < width; i++) {
        ans.add(map.get(i));
    }
    return ans;
}
```

```java
public static class verticalPair {
    TreeNode node = null;
    int hl = 0; // horizontal Level

    verticalPair(TreeNode node, int hl) {
        this.node = node;
        this.hl = hl;
    }
}

// ans = {minHL,maxHL}
public static void width(TreeNode root, int hl, int[] ans) {
    if (root == null)
        return;

    ans[0] = Math.min(hl, ans[0]);
    ans[1] = Math.max(hl, ans[1]);

    width(root.left, hl - 1, ans);
    width(root.right, hl + 1, ans);
}

public static ArrayList<Integer> BottomView(TreeNode root) {
    LinkedList<verticalPair> que = new LinkedList<>();

    int[] minMax = new int[2];
    width(root, 0, minMax);
    int length = minMax[1] - minMax[0] + 1;
    ArrayList<Integer> ans = new ArrayList<>();
    for (int i = 0; i < length; i++)
        ans.add(0);

    que.addLast(new verticalPair(root, -minMax[0]));

    while (que.size() != 0) {
        int size = que.size();
        while (size-- > 0) {
            verticalPair rp = que.removeFirst();

            ans.set(rp.hl, rp.node.val);

            if (rp.node.left != null)
                que.addLast(new verticalPair(rp.node.left, rp.hl - 1));

            if (rp.node.right != null)
                que.addLast(new verticalPair(rp.node.right, rp.hl + 1));
        }
    }

    return ans;
}

// input section-----------------------------------------------------
```

only change this
just overrige





```java
public static class verticalPair {
    TreeNode node = null;
    int hl = 0; // horizontal Level

    verticalPair(TreeNode node, int hl) {
        this.node = node;
        this.hl = hl;
    }
}

// ans = {minHL,maxHL}
public static void width(TreeNode root, int hl, int[] ans) {
    if (root == null)
        return;

    ans[0] = Math.min(hl, ans[0]);
    ans[1] = Math.max(hl, ans[1]);

    width(root.left, hl - 1, ans);
    width(root.right, hl + 1, ans);
}

public static ArrayList<Integer> TopView(TreeNode root) {
    LinkedList<verticalPair> que = new LinkedList<>();

    int[] minMax = new int[2];
    width(root, 0, minMax);
    int length = minMax[1] - minMax[0] + 1;
    ArrayList<Integer> ans = new ArrayList<>();
    for (int i = 0; i < length; i++)
        ans.add((int) 1e9);

    que.addLast(new verticalPair(root, -minMax[0]));

    while (que.size() != 0) {
        int size = que.size();
        while (size-- > 0) {
            verticalPair rp = que.removeFirst();

            if (ans.get(rp.hl) == (int) 1e9)
                ans.set(rp.hl, rp.node.val);

            if (rp.node.left != null)
                que.addLast(new verticalPair(rp.node.left, rp.hl - 1));

            if (rp.node.right != null)
                que.addLast(new verticalPair(rp.node.right, rp.hl + 1));
        }
    }

    return ans;
}
```

backward diogonal

```
    public static ArrayList<ArrayList<Integer>> diagonalOrder
(TreeNode root) {
        LinkedList<TreeNode> que = new LinkedList<>();
        ArrayList<ArrayList<Integer>> ans = new ArrayList<>();

        que.addLast(root);
        while(que.size() != 0){  // diagonal

            int size = que.size();
            ArrayList<Integer> smallAns = new ArrayList<>();

            while(size-->0){   // help to traverse each Component of
that particular diagonal.
                TreeNode rn = que.removeFirst();
                while(rn != null){   // traverse a component.
                 smallAns.add(rn.val);
                  if(rn.left != null) que.addLast(rn.left);

                    rn = rn.right;
                }
             }

             ans.add(smallAns);
        }

        return ans;
    }
```

forward diogonal

```
    public static ArrayList<ArrayList<Integer>> diagonalOrder
(TreeNode root) {
        LinkedList<TreeNode> que = new LinkedList<>();
        ArrayList<ArrayList<Integer>> ans = new ArrayList<>();

        que.addLast(root);
        while(que.size() != 0){ // help to traverse each diagonal
            int size = que.size();
            ArrayList<Integer> smallAns = new ArrayList<>();

            while(size-->0){ // help to traverse all components.

                TreeNode rn = que.removeFirst();
                while(rn != null){ // help to traverse each component.
                    smallAns.add(rn.val);
                    if(rn.right != null) que.addLast(rn.right);

                    rn = rn.left;
                }
            }

            ans.add(smallAns);

        }
        return ans;
```

```java
public static class verticalPair {
    TreeNode node = null;
    int hl = 0; // horizontal Level

    verticalPair(TreeNode node, int hl) {
        this.node = node;
        this.hl = hl;
    }
}

// ans = {minHL,maxHL}
public static void width(TreeNode root, int hl, int[] ans) {
    if (root == null)
        return;

    ans[0] = Math.min(hl, ans[0]);
    ans[1] = Math.max(hl, ans[1]);

    width(root.left, hl - 1, ans);
    width(root.right, hl + 1, ans);
}

public static ArrayList<Integer> verticalOrderSum(TreeNode root) {
    LinkedList<verticalPair> que = new LinkedList<>();

    int[] minMax = new int[2];
    width(root, 0, minMax);
    int length = minMax[1] - minMax[0] + 1;
    ArrayList<Integer> ans = new ArrayList<>();
    for (int i = 0; i < length; i++)
        ans.add(0);

    que.addLast(new verticalPair(root, -minMax[0]));

    while (que.size() != 0) {
        int size = que.size();
        while (size-- > 0) {
            verticalPair rp = que.removeFirst();

            ans.set(rp.hl, ans.get(rp.hl) + rp.node.val);

            if (rp.node.left != null)
                que.addLast(new verticalPair(rp.node.left, rp.hl - 1));

            if (rp.node.right != null)
                que.addLast(new verticalPair(rp.node.right, rp.hl + 1));
        }
    }

    return ans;
}
```

```java
public static void dfs(TreeNode root,int hl,ArrayList<Integer>
ans){
    if(root == null) return;

    ans.set(hl, ans.get(hl) + root.val);

    dfs(root.left, hl - 1, ans);
    dfs(root.right, hl + 1, ans);
}

public static ArrayList<Integer> verticalOrderSum(TreeNode root){

    int[] minMax = new int[2];
    width(root, 0 , minMax);

    int len = minMax[1] - minMax[0] + 1;

    ArrayList<Integer> ans = new ArrayList<>();
    for(int i = 0; i < len;i++) ans.add(0);

    dfs(root, Math.abs(minMax[0]), ans);
    return ans;
}
```

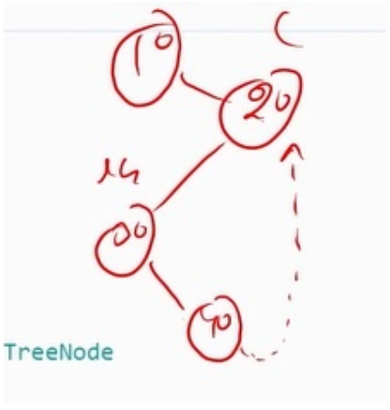
```java
public static ArrayList<Integer> diagonalOrderSum(TreeNode root) {
  LinkedList<TreeNode> que = new LinkedList<>();
  ArrayList<Integer> ans = new ArrayList<>();
  que.addLast(root);
  while(que.size() != 0){
      int size = que.size();
      int sum = 0;
      while(size-->0){
          TreeNode rn = que.removeFirst();
          while(rn != null){
              sum += rn.val;
              if(rn.left != null) que.addLast(rn.left);

              rn = rn.right;
          }
      }

      ans.add(sum);
  }
  return ans;
}
```

```java
  }

  public static void dfs(TreeNode node,int diagNo,ArrayList<Integer> ans){
      if(node == null) return;

      if(diagNo == ans.size()) ans.add(0);
      ans.set(diagNo, ans.get(diagNo) + node.val);

      dfs(node.left, diagNo + 1, ans);
      dfs(node.right, diagNo, ans);
  }

  public static ArrayList<Integer> diagonalOrderSum(TreeNode root) {
      ArrayList<Integer> ans = new ArrayList<Integer>();
      dfs(root, 0, ans);
    return ans;
  }
```

```java
public static ArrayList<TreeNode> nodeToRootPath(TreeNode root, int data, ArrayList<TreeNode> path) {
 if (root == null)
   return new ArrayList<>();
 if (root.val == data) {

  ArrayList<TreeNode> ans = new ArrayList<>();
   ans.add(root);
   return ans;
  }

 ArrayList<TreeNode> left= nodeToRootPath(root.left, data, path);
 if(left.size()>0)
{
    left.add(root);
    return left;
}
  ArrayList<TreeNode> right =nodeToRootPath(root.right, data, path);
  if(right.size()>0){

     right.add(root);
     return right;
  }


  return new ArrayList<>();
}

public static ArrayList<TreeNode> nodeToRootPath(TreeNode root, int data) {
 ArrayList<TreeNode> path = new ArrayList<>();
  return nodeToRootPath(root, data, path);

}
// input section=====================================================
```

```java

public static boolean nodeToRootPath(TreeNode root, int data, ArrayList<TreeNode> path) {
  if (root == null)
    return false;
  if (root.val == data) {
    path.add(root);
    return true;
  }

  boolean res = nodeToRootPath(root.left, data, path) || nodeToRootPath(root.right, data, path);
  if (res)
    path.add(root);

  return res;
}

public static ArrayList<TreeNode> nodeToRootPath(TreeNode root, int data) {
  ArrayList<TreeNode> path = new ArrayList<>();
  nodeToRootPath(root, data, path);
  return path;
}
```

```java
    public static TreeNode getRightMostNode(TreeNode leftNode,TreeNode
curr){

        while(leftNode.right != null && leftNode.right != curr){
            leftNode = leftNode.right;
        }

        return leftNode;

    }
```



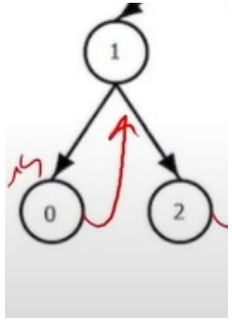
40 is extreme right
whose right can be either null whihc means
there is no thread
or it is pointing to currentt node

```java
public static ArrayList<Integer> morrisInTraversal(TreeNode node) {
    ArrayList<Integer> ans = new ArrayList<>();
    TreeNode curr = node;

    while(curr != null){
        TreeNode leftNode = curr.left;
        if(leftNode == null){
            ans.add(curr.val);
            curr = curr.right;
        }else{

            TreeNode rightMostNode = getRightMostNode(leftNode,curr
);
            if(rightMostNode.right == null){  // threade create
                rightMostNode.right = curr;
                curr = curr.left;
            }else{   // threade destroy
                rightMostNode.right = null;
                ans.add(curr.val);
                curr = curr.right;
            }
        }
    }

    return ans;
```

wne cueent last node pe pahunch
jayega get out

if left node is null then
prinit and go to right
similar to leftcall, print and right - > inorder traversal

run scenerioa for 0



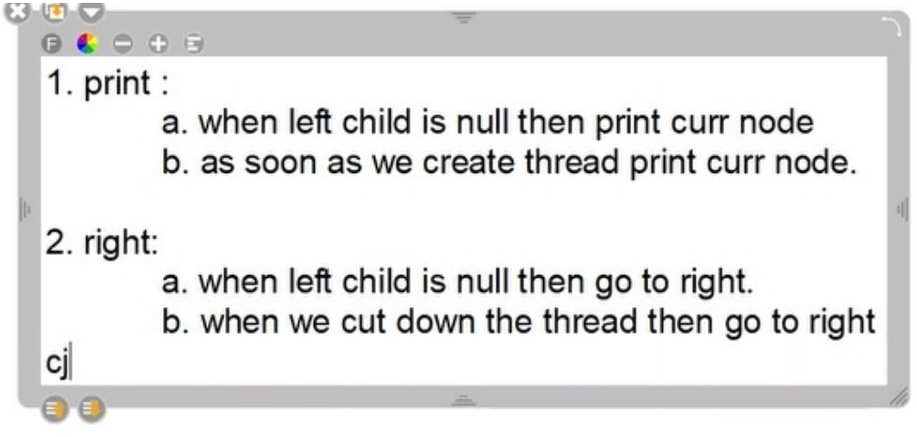
agar left null nhi he ,

left node ka extreme right leke aao
here there is 2 scenerio-
extreme right ka right is eitther null or pointing to current node
1. if null then , create thread and goto left
2. if not null , then remove thread (current ka left is processed), print and got to right

1. print :
    a. if left is null
    b. if threade is cut down then print current node
2. Mark of left subtree is completely processed or not :
    if threade already exists

1. right -> if left not exist or either left subtree is completely processed

Pre Order Morris Traversal In Binary Tree | Using O(1) Space | Pepcoding Solution in Hindi




```java
public static ArrayList<Integer> morrisPreTraversal(TreeNode root)
{
    ArrayList<Integer> ans = new ArrayList<>();
    TreeNode curr = root;
    while(curr != null){
        TreeNode leftNode = curr.left;
        if(leftNode == null){
            ans.add(curr.val);
            curr = curr.right;
        }else{
            TreeNode rightMostNode = getRightMostNode(leftNode,curr
);

            if(rightMostNode.right == null){ // thread have to
create
                ans.add(curr.val);
                rightMostNode.right = curr; // thread creation
                curr = curr.left;
            }else{   // thread is present
                rightMostNode.right = null;   // thread cut down
                curr = curr.right;
            }
        }
    }

    return ans;
}
```

```java
public static TreeNode getRightMostNode(TreeNode rightMost,TreeNode
curr){

    while(rightMost.right != null && rightMost.right != curr)
        rightMost = rightMost.right;

    return rightMost;
}
```

Binary Search Tree Iterator Using Stack | Using Log(N) Space | Leetcode 173 Solution in Hindi

```java
public static class BSTIterator {
    LinkedList<TreeNode> st; // addFirst, removeFirst

    public BSTIterator(TreeNode root) {
        this.st = new LinkedList<>();
        addAllLeft(root);
    }

    private void addAllLeft(TreeNode node) {
        while (node != null) {
            this.st.addFirst(node);
            node = node.left;
        }
    }

    public int next() {
        TreeNode topNode = this.st.removeFirst();

        addAllLeft(topNode.right);

        return topNode.val;
    }

    public boolean hasNext() {
        return this.st.size() != 0;
    }
}
```

Binary Tree Iterator Using Morris Traversal | Using O(1) Space | Leetcode 173 Solution In Hindi

```java
public static class BSTIterator {
    TreeNode curr = null;

    public BSTIterator(TreeNode root) {
        this.curr = root;
    }

    // rmn : right most node
    private TreeNode getRightMostNode(TreeNode rmn) {
        while (rmn.right != null && rmn.right != this.curr)
            rmn = rmn.right;
        return rmn;
    }
```

```java
private TreeNode morrisTraversal() {
    TreeNode res = null;

    while (this.curr != null) {
        TreeNode leftNode = this.curr.left;
        if (leftNode == null) {
            res = this.curr;
            this.curr = this.curr.right;
            break;
        } else {
            TreeNode rightMostNode = getRightMostNode(leftNode);
            if (rightMostNode.right == null) {
                rightMostNode.right = this.curr;// thread creation
                this.curr = this.curr.left;
            } else {
                res = this.curr;
                rightMostNode.right = null; // thread cut down
                this.curr = this.curr.right;
                break;
            }
        }
    }

    return res;
}
```

```java
    public int next() {
        TreeNode res = morrisTraversal();
        return res.val;

    }

    public boolean hasNext() {
        return this.curr != null;

    }
```

Root To All Leaf Path In Binary Tree

All Single Child Parent In Binary Tree

```java
  public static void exactlyOneChild(TreeNode root,ArrayList<Integer>
ans){

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

      if(root.left == null || root.right == null){
          ans.add(root.val);
      }

      exactlyOneChild(root.left,ans);
      exactlyOneChild(root.right,ans);
  }

  public static ArrayList<Integer> exactlyOneChild(TreeNode root) {
    ArrayList<Integer> ans = new ArrayList<>();
    exactlyOneChild(root,ans);
    return ans;
  }
```

Count All Single Child Parent In Binary Tree