

▼ Medium Problems

- J [Bottom View of Binary Tree.java](#)
- J [Boundary Traversal of Binary Tree.java](#)
- J [Check if the Binary tree is height-balanced or not.java](#)
- J [Check if two trees are identical or not.java](#)
- J [Diameter of Binary Tree.java](#)
- J [Height of a Binary Tree.java](#)
- J [Maximum path sum.java](#)
- J [Right or Left View of Binary Tree.java](#)
- J [Symmetric Binary Tree.java](#)
- J [Top View of Binary Tree.java](#)
- J [Vertical Order Traversal of Binary Tree.java](#)
- J [Zig Zag Traversal of Binary Tree.java](#)

```

1  /*
2  |   Bottom View of Binary Tree
3  |   Given a binary tree, print the bottom view from left to right.
4  |   A node is included in bottom view if it can be seen when we look at the tree from bottom.
5  |
6  |   | | | | | | | | | | 20
7  |   | | | | | | | | | / \
8  |   | | | | | | | | 8   22
9  |   | | | | | | | | / \
10 |   | | | | | | | 5   3   25
11 |   | | | | | | | / \
12 |   | | | | | | 10  14
13 |
14 |   For the above tree, the bottom view is 5 10 3 14 25.
15 |   If there are multiple bottom-most nodes for a horizontal distance from root, then
16 |   print the later one in level traversal. For example, in the below diagram, 3 and
17 |   4 are both the bottommost nodes at horizontal distance 0, we need to print 4.
18 |
19 |   | | | | | | | | | | 20
20 |   | | | | | | | | | / \
21 |   | | | | | | | | 8   22
22 |   | | | | | | | | / \
23 |   | | | | | | | 5   3 4   25
24 |   | | | | | | | / \
25 |   | | | | | | 10  14
26 |
27 |   For the above tree the output should be 5 10 4 14 25.
28 |
29 |   Example 1:
30 |
31 |   Input:
32 |   | | | 1
33 |   | / \
34 |   3   2
35 |   Output: 3 1 2
36 |   Explanation:
37 |   First case represents a tree with 3 nodes
38 |   and 2 edges where root is 1, left child of
39 |   1 is 3 and right child of 1 is 2.
40 |
41 |   Thus nodes of the binary tree will be
42 |   printed as such 3 1 2.
43 |   Example 2:
44 |
45 |   Input:
46 |   | | | | 10
47 |   | / \
48 |   20  30
49 |   / \
50 |  40  60
51 |   Output: 40 20 60 30
52 |
53 | */

```

Bottom_View_of_Binary_Tree

```

60  Codeium: Refactor | Explain
61  class Node{
62  |   int data;
63  |   Node left;
64  |   Node right;
65  |   Node(int key){
66  |   |   this.data = key;
67  |   }
68  }
69  Codeium: Refactor | Explain
70  class Pair{
71  |   Node node;
72  |   int hd;
73  |   Pair(Node node, int hd){
74  |   |   this.node = node;
75  |   |   this.hd = hd;
76  |   }
77  }

```

77

Codeium: Refactor | Explain

78 `public class Bottom_View_of_Binary_Tree {`

Codeium: Refactor | Explain | Generate Javadoc | X

79 `public static ArrayList<Integer> bottomView(Node root){`80 `ArrayList<Integer> ans = new ArrayList<>();`81 `TreeMap<Integer, Integer> map = new TreeMap<>();`82 `if(root == null) return ans;`83 `Queue<Pair> q = new LinkedList<>();`84 `q.add(new Pair(root, 0));`85 `while(!q.isEmpty()){`86 `Pair it = q.remove();`87 `int hd = it.hd;`88 `Node temp = it.node;`89 `map.put(hd, temp.data);`90 `if(temp.left != null) q.add(new Pair(temp.left, hd - 1));`91 `if(temp.right != null) q.add(new Pair(temp.right, hd + 1));`92 `}`93 `for(Integer num : map.values()){`94 `ans.add(num);`95 `}`96 `return ans;`97 `}`

Codeium: Refactor | Explain | Generate Javadoc | X

98 `public static void main(String[] args) {`99 `/*`100 `Real World Representation the below tree into the code.`

```

101         |   |   |   |   |
102         |   |   |   |   |
103         |   |   |   |   |
104         |   |   |   |   |
105         |   |   |   |   |
106         |   |   |   |   |
107         |   |   |   |   |
108         |   |   |   |   |
109         |   |   |   |   |
110         |   |   |   |   |
111         |   |   |   |   |
112         |   |   |   |   |
113         |   |   |   |   |
114         |   |   |   |   |
115         |   |   |   |   |
116         |   |   |   |   |
117         |   |   |   |   |
118         |   |   |   |   |
119         |   |   |   |   |
120         |   |   |   |   |
121         |   |   |   |   |
122         |   |   |   |   |

```

108 `*/`109 `Node root = new Node(1);`110 `root.left = new Node(2);`111 `root.right = new Node(3);`112 `root.left.left = new Node(4);`113 `root.left.right = new Node(5);`114 `root.left.right.left = new Node(8);`115 `root.right.left = new Node(6);`116 `root.right.right = new Node(7);`117 `root.right.right.left = new Node(9);`118 `root.right.right.right = new Node(10);`119 `System.out.println(bottomView(root));`120 `}`121 `}`

122

> Task :Bottom_View_of_Binary_Tree.main()

[4, 8, 6, 9, 7, 10]

Boundary_Traversal_of_Binary_Tree

```
public class Boundary_Traversal_of_Binary_Tree {
    public static boolean isLeaf(Node node){
        return node.left == null && node.right == null;
    }
    public static void addLeftBoundry(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;
        }
    }
    public static void addRightBoundry(Node root, ArrayList<Integer> res){
        ArrayList<Integer> temp = new ArrayList<>();
        Node curr = root.right;
        while(curr != null){
            if(isLeaf(curr) == false) temp.add(curr.data);
            if(curr.right != null) curr = curr.right;
            else curr = curr.left;
        }
        // Adding Right boundary in reverse order to Achieve Anti-ClockWise
        for(int i = temp.size() - 1; i >= 0; i--){
            res.add(temp.get(i));
        }
    }
    public static void addLeaves(Node root, ArrayList<Integer> res){
        if(isLeaf(root)){
            res.add(root.data);
        }
        if(root.left != null) addLeaves(root.left, res);
        if(root.right != null) addLeaves(root.right, res);
    }
    public static ArrayList<Integer> printBoundry(Node root){
        ArrayList<Integer> ans = new ArrayList<>();
        if(isLeaf(root) == false) ans.add(root.data);
        addLeftBoundry(root, ans);
        addLeaves(root, ans);
        addRightBoundry(root, ans);
        return ans;
    }
}
```

```

public static void main(String[] args) {
    /*
        Real World Representation the below tree into the code.
        1
       / \
      2   3
     / \ / \
    4  5 6  7
     / \ / \
    8  9 10
    */
    Node root = new Node(1);
    root.left = new Node(2);
    root.right = new Node(3);
    root.left.left = new Node(4);
    root.left.right = new Node(5);
    root.left.right.left = new Node(8);
    root.right.left = new Node(6);
    root.right.right = new Node(7);
    root.right.right.left = new Node(9);
    root.right.right.right = new Node(10);
    System.out.println(printBoundary(root));
}
}

```

```

> Task :Boundary_Traversal_of_Binary_Tree.main()
[1, 2, 4, 8, 6, 9, 10, 7, 3]

```

Check_if_the_Binary_tree_is_height_balanced_or_not

```

1  /*
2     110. Balanced Binary Tree
3     Given a binary tree, determine if it is
4     height-balanced
5
6     Example 1:
7     Input: root = [3,9,20,null,null,15,7]
8     Output: true
9
10    Example 2:
11    Input: root = [1,2,2,3,3,null,null,4,4]
12    Output: false
13    Example 3:
14
15    Input: root = []
16    Output: true
17 */

```

```

28 public class Check_if_the_Binary_tree_is_height_balanced_or_not {
29     public static int maxDepth(Node root) {
30         if(root == null) return 0;
31         int lh = maxDepth(root.left);
32         if(lh == -1) return -1;
33         int rh = maxDepth(root.right);
34         if(rh == -1) return -1;
35         if(Math.abs(lh - rh) > 1) return -1;
36         return 1 + Math.max(lh, rh);
37     }
38     public static boolean isBalanced(Node root) {
39         return maxDepth(root) != -1;
40     }
41     public static void main(String[] args) {
42         /*
43         Real World Representation the below tree into the code.
44             1
45          /  \
46         2    3
47        / \  / \
48       4  5 6  7
49        /  / \ \
50       8 9  10
51         */
52         Node root = new Node(1);
53         root.left = new Node(2);
54         root.right = new Node(3);
55         root.left.left = new Node(4);
56         root.left.right = new Node(5);
57         root.left.right.left = new Node(8);
58         root.right.left = new Node(6);
59         root.right.right = new Node(7);
60         root.right.right.left = new Node(9);
61         root.right.right.right = new Node(10);
62         System.out.println(isBalanced(root));
63     }
64 }

```

> Task :Check_if_the_Binary_tree_is_height_balanced_or_not.main()
true

Check if two trees are identical or not

```
/*
100. Same Tree
Given the roots of two binary trees p and q, write a function to check if they are the same or not.

Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.

Example 1:
Input: p = [1,2,3], q = [1,2,3]
Output: true
Example 2:

Input: p = [1,2], q = [1,null,2]
Output: false

Example 3:
Input: p = [1,2,1], q = [1,1,2]
Output: false
*/
```

```
30 public class Check_if_two_trees_are_identical_or_not {
31     public static boolean isSameTree(Node p, Node q) {
32         if(p == null || q == null){
33             return p == q;
34         }
35         return p.data == q.data && isSameTree(p.left, q.left) && isSameTree(p.right, q.right);
36     }
37     public static void main(String[] args) {
38         /*
39             Real World Representation the below tree into the code.
40             |
41             | 1
42             / \
43            2   3
44           / \ / \
45          4  5 6  7
46         / \ / \ \
47        8  9 10
48         */
49         // First Tree
50         Node root = new Node(1);
51         root.left = new Node(2);
52         root.right = new Node(3);
53         root.left.left = new Node(4);
54         root.left.right = new Node(5);
55         root.left.right.left = new Node(8);
56         root.right.left = new Node(6);
57         root.right.right = new Node(7);
58         root.right.right.left = new Node(9);
59         root.right.right.right = new Node(10);
60
61         // Second Tree
62         Node root2 = new Node(1);
63         root2.left = new Node(2);
64         root2.right = new Node(3);
65         root2.left.left = new Node(4);
66         root2.left.right = new Node(5);
67         root2.left.right.left = new Node(8);
68         root2.right.left = new Node(6);
69         root2.right.right = new Node(7);
70         root2.right.right.left = new Node(9);
71         root2.right.right.right = new Node(10);
72
73         System.out.println(isSameTree(root, root2));
74     }
75 }
76
```

```
> Task :Check_if_two_trees_are_identical_or_not.main()
true
```

Diameter_of_Binary_Tree

```
/*
543. Diameter of Binary Tree
Given the root of a binary tree, return the length of the diameter of the tree.

The diameter of a binary tree is the length of the longest path between any two nodes in a tree. This path may or may not pass through the root.

The length of a path between two nodes is represented by the number of edges between them.

Example 1:
Input: root = [1,2,3,4,5]
Output: 3
Explanation: 3 is the length of the path [4,2,1,3] or [5,2,1,3].

Example 2:
Input: root = [1,2]
Output: 1
*/
```

```
27
28 public class Diameter_of_Binary_Tree {
29     public static int height(Node root, int[] diameter){
30         if(root == null) return 0;
31
32         int lh = height(root.left, diameter);
33         int rh = height(root.right, diameter);
34         diameter[0] = Math.max(diameter[0], lh + rh);
35         return 1 + Math.max(lh, rh);
36     }
37     public static int diameterOfBinaryTree(Node root) {
38         int[] diameter = new int[1];
39         height(root, diameter);
40         return diameter[0];
41     }
42     public static void main(String[] args) {
43         /*
44         Real World Representation the below tree into the code.
45         |
46         | 1
47         | / \
48         | 2   3
49         | / \ / \
50         | 4 5 6 7
51         | /   / \
52         | 8   9 10
53         */
54         Node root = new Node(1);
55         root.left = new Node(2);
56         root.right = new Node(3);
57         root.left.left = new Node(4);
58         root.left.right = new Node(5);
59         root.left.right.left = new Node(8);
60         root.right.left = new Node(6);
61         root.right.right = new Node(7);
62         root.right.right.left = new Node(9);
63         root.right.right.right = new Node(10);
64         System.out.println(diameterOfBinaryTree(root));
65     }
66 }
```

> Task :Diameter_of_Binary_Tree.main()

6

Height_of_a_Binary_Tree

```
1  /*
2   104. Maximum Depth of Binary Tree
3   Given the root of a binary tree, return its maximum depth.
4
5   A binary tree's maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.
6
7   Example 1:
8
9
10  Input: root = [3,9,20,null,null,15,7]
11  Output: 3
12  Example 2:
13
14  Input: root = [1,null,2]
15  Output: 2
16  */
```

```

27 public class Height_of_a_Binary_Tree {
28     // Iterative
29     /*
30     public int maxDepth(TreeNode root) {
31         int height = 0;
32         Queue queue = new LinkedList<>();
33         if(root == null) return 0;
34         queue.add(root);
35         while(!queue.isEmpty()){
36             int k = queue.size();
37             for(int i = 0; i < k; i++){
38                 if(queue.peek().left != null) queue.add(queue.peek().left);
39                 if(queue.peek().right != null) queue.add(queue.peek().right);
40                 queue.remove();
41             }
42             height++;
43         }
44         return height;
45     }
46     */
47
48     // Recurive
49     public static int maxDepth(Node root) {
50         if(root == null) return 0;
51         int lh = maxDepth(root.left);
52         int rh = maxDepth(root.right);
53         return 1 + Math.max(lh, rh);
54     }
55     public static void main(String[] args) {
56         /*
57         Real World Representation the below tree into the code.
58         |
59         | 1
60         | / \
61         | 2   3
62         | / \ / \
63         | 4 5 6 7
64         | /   / \
65         | 8   9 10
66         */
67         Node root = new Node(1);
68         root.left = new Node(2);
69         root.right = new Node(3);
70         root.left.left = new Node(4);
71         root.left.right = new Node(5);
72         root.left.right.left = new Node(8);
73         root.right.left = new Node(6);
74         root.right.right = new Node(7);
75         root.right.right.left = new Node(9);
76         root.right.right.right = new Node(10);
77         System.out.println(maxDepth(root));
78     }
}

```

> Task :Height_of_a_Binary_Tree.main()

Maximum_path_sum

```

1  /*
2  124. Binary Tree Maximum Path Sum
3  A path in a binary tree is a sequence of nodes where each pair of adjacent nodes in the sequence has an edge connecting them.
4  A node can only appear in the sequence at most once.
5  Note that the path does not need to pass through the root.
6
7  The path sum of a path is the sum of the node's values in the path.
8
9  Given the root of a binary tree, return the maximum path sum of any non-empty path.
10
11 Example 1:
12 Input: root = [1,2,3]
13 Output: 6
14 Explanation: The optimal path is 2 -> 1 -> 3 with a path sum of 2 + 1 + 3 = 6.
15
16 Example 2:
17 Input: root = [-10,9,20,null,null,15,7]
18 Output: 42
19 Explanation: The optimal path is 15 -> 20 -> 7 with a path sum of 15 + 20 + 7 = 42.
20 */
--
31 public class Maximum_path_sum {
32     public static int maximumPath(Node root, int[] maxSum){
33         if(root == null) return 0;
34         // Not Consider Negative Value Take "0" Instead
35         int lSum = Math.max(0, maximumPath(root.left, maxSum));
36         // Not Consider Negative Value Take "0" Instead
37         int rSum = Math.max(0, maximumPath(root.right, maxSum));
38         maxSum[0] = Math.max(maxSum[0], root.data + (lSum + rSum));
39         return root.data + Math.max(lSum, rSum);
40     }
41     public static int maxPathSum(Node root) {
42         int[] maxSum = new int[1];
43         maxSum[0] = Integer.MIN_VALUE;
44         maximumPath(root, maxSum);
45         return maxSum[0];
46     }
47     public static void main(String[] args) {
48         /*
49         Real World Representation the below tree into the code.
50
51         1
52        / \
53       2   3
54      / \ / \
55     4  5 6  7
56    / \ / \ \
57   8  9 10
58
59         */
60         Node root = new Node(1);
61         root.left = new Node(2);
62         root.right = new Node(3);
63         root.left.left = new Node(4);
64         root.left.right = new Node(5);
65         root.left.right.left = new Node(8);
66         root.right.left = new Node(6);
67         root.right.right = new Node(7);
68         root.right.right.left = new Node(9);
69         root.right.right.right = new Node(10);
70         System.out.println(maxPathSum(root));
71     }
72 }

```

> Task :Maximum_path_sum.main()

Binary_Tree_Right_Side_View

```
1  /*
2  199. Binary Tree Right Side View
3  Given the root of a binary tree, imagine yourself standing on the right side of it, return the values of the nodes you
4  can see ordered from top to bottom.
5
6  Example 1:
7  Input: root = [1,2,3,null,5,null,4]
8  Output: [1,3,4]
9
10 Example 2:
11 Input: root = [1,null,3]
12 Output: [1,3]
13
14 Example 3:
15 Input: root = []
16 Output: []
17 */
18
```

```
33 public class Binary_Tree_Right_Side_View {
34     /*
35     // Iterative
36     public static List<Integer> rightSideView(Node root) {
37         List<Integer> ans = new ArrayList<>();
38         Queue<Node> q = new LinkedList<>();
39         if(root == null) return ans;
40         q.add(root);
41         while(!q.isEmpty()){
42             int k = q.size();
43             int RightMost = 0;
44             for(int i = 0; i < k; i++){
45                 if(q.peek().left != null) q.add(q.peek().left);
46                 if(q.peek().right != null) q.add(q.peek().right);
47                 RightMost = q.remove().data;
48             }
49             ans.add(RightMost);
50         }
51         return ans;
52     }
53     */
54
55     // Recursive
56     public static List<Integer> rightSideView(Node root) {
57         List<Integer> ans = new ArrayList<>();
58         rightView(root, ans, 0);
59         return ans;
60     }
61     public static void rightView(Node root, List<Integer> ans, int currDepth){
62         if(root == null) return;
63         // Storing ans in our list
64         if(currDepth == ans.size()) ans.add(root.data);
65         // Right
66         if(root.right != null) rightView(root.right, ans, currDepth + 1);
67         // Left
68         if(root.left != null) rightView(root.left, ans, currDepth + 1);
69     }
70 }
```

```

70 public static void main(String[] args) {
71     /*
72         Visualisation of the tree in real world
73         1
74         / \
75        2   3
76       / \ / \
77      4  5 6  7
78       /  \ / \
79      8   9 10
80     */
81     Node root = new Node(1);
82     root.left = new Node(2);
83     root.right = new Node(3);
84     root.left.left = new Node(4);
85     root.left.right = new Node(5);
86     root.left.right.left = new Node(8);
87     root.right.left = new Node(6);
88     root.right.right = new Node(7);
89     root.right.right.left = new Node(9);
90     root.right.right.right = new Node(10);
91     System.out.println(rightSideView(root));
92 }
93 }

```

```

> Task :Binary_Tree_Right_Side_View.main()
[1, 3, 7, 10]

```

Symmetric_Binary_Tree

11 Problems / Symmetric Binary Tree.java

```
/*
101. Symmetric Tree
Given the root of a binary tree, check whether it is a mirror of itself (i.e., symmetric around its center).

Example 1:
Input: root = [1,2,2,3,4,4,3]
Output: true

Example 2:
Input: root = [1,2,2,null,3,null,3]
Output: false
*/
```

```
23 public class Symmetric_Binary_Tree {
    Codeium: Refactor | Explain | Generate Javadoc | X
24     public static boolean isSymmetric(Node root) {
25         return root == null || isSymmetricHelp(root.left, root.right);
26     }
    Codeium: Refactor | Explain | Generate Javadoc | X
27     public static boolean isSymmetricHelp(Node left, Node right){
28         if(left == null || right == null){
29             return left == right;
30         }
31         if(left.data != right.data) return false;
32         return isSymmetricHelp(left.left, right.right) && isSymmetricHelp(left.right, right.left);
33     }
    Codeium: Refactor | Explain | Generate Javadoc | X
34     public static void main(String[] args) {
35         /*
36          Visualisation of the tree in real world
37          |
38          |   |   |   |   |
39          |   / \ / \ / \
40          | 2   3 4   5 6   7
41          | / \ / \ / \
42          | 4 5 6 7 8   9 10
43          |
44         */
45         Node root = new Node(1);
46         root.left = new Node(2);
47         root.right = new Node(2);
48         root.left.left = new Node(3);
49         root.left.right = new Node(4);
50         root.right.left = new Node(4);
51         root.right.right = new Node(3);
52         System.out.println(isSymmetric(root));
53     }
54 }
55
```

> Task :Symmetric_Binary_Tree.main()

true

Top_View_of_Binary_Tree

```
/*
Q. Top View of Binary Tree
Given below is a binary tree. The task is to print the top view of binary tree. Top view of a binary tree is the set of nodes
visible when the tree is viewed from the top. For the given below tree

      1
     / \
    2   3
   / \ / \
  4  5 6  7

Top view will be: 4 2 1 3 7
Note: Return nodes from leftmost node to rightmost node. Also if 2 nodes are outside the shadow of the tree and are at same position
then consider the extreme ones only(i.e. leftmost and rightmost).
For ex - 1 2 3 N 4 5 N 6 N 7 N 8 N 9 N N N N N will give 8 2 1 3 as answer.
Here 8 and 9 are on the same position but 9 will get shadowed.

Example 1:

Input:
      1
     / \
    2   3
Output: 2 1 3
Example 2:

Input:
      10
     /  \
    20   30
   / \  / \
  40 60 90 100
Output: 40 20 10 30 100
*/
```

Codeium: Refactor | Explain

```
class Pair{
    int hd;
    Node node;
    Pair(int hd, Node node){
        this.hd = hd;
        this.node = node;
    }
}
```

Codeium: Refactor | Explain

```
42 class Node{
43     int data;
44     Node left;
45     Node right;
46     Node(int key){
47         this.data = key;
48     }
49 }
50
```

Codeium: Refactor | Explain

```
60 public class Top_View_of_Binary_Tree {
    Codeium: Refactor | Explain | Generate Javadoc | X
61     public static ArrayList<Integer> topView(Node root) {
62         ArrayList<Integer> ans = new ArrayList<>();
63         if(root == null) return ans;
64         Map<Integer, Integer> map = new TreeMap<>();
65         Queue<Pair> q = new LinkedList<>();
66         q.add(new Pair(0, root));
67         while(!q.isEmpty()){
68             Pair it = q.remove();
69             int hd = it.hd;
70             Node temp = it.node;
71             if(map.get(hd) == null){
72                 map.put(hd, temp.data);
73             }
74             if(temp.left != null){
75                 q.add(new Pair(hd - 1, temp.left));
76             }
77             if(temp.right != null){
78                 q.add(new Pair(hd + 1, temp.right));
79             }
80         }
81         for(Map.Entry<Integer, Integer> entry: map.entrySet()){
82             ans.add(entry.getValue());
83         }
84         return ans;
85     }
    Codeium: Refactor | Explain | Generate Javadoc | X
```



```

86 public static void main(String[] args) {
87     /*
88         Real World Representation the below tree into the code.
89         |
90         |   /   \
91         | 2     3
92         / \   / \
93        4  5 6   7
94         /   \   / \
95        8     9   10
96     */
97     Node root = new Node(1);
98     root.left = new Node(2);
99     root.right = new Node(3);
100    root.left.left = new Node(4);
101    root.left.right = new Node(5);
102    root.left.right.left = new Node(8);
103    root.right.left = new Node(6);
104    root.right.right = new Node(7);
105    root.right.right.left = new Node(9);
106    root.right.right.right = new Node(10);
107    System.out.println(topView(root));
108 }
109 }
110

```

```

> Task :Top_View_of_Binary_Tree.main()
[4, 2, 1, 3, 7, 10]

```

Vertical Order Traversal of a Binary Tree

```
1  /*
2  987. Vertical Order Traversal of a Binary Tree
3  Given the root of a binary tree, calculate the vertical order traversal of the binary tree.
4
5  For each node at position (row, col), its left and right children will be at positions (row + 1, col - 1) and
6  (row + 1, col + 1) respectively. The root of the tree is at (0, 0).
7
8  The vertical order traversal of a binary tree is a list of top-to-bottom orderings for each
9  column index starting from the leftmost column and ending on the rightmost column. There may be multiple nodes in the
10 same row and same column.
11 In such a case, sort these nodes by their values.
12
13 Return the vertical order traversal of the binary tree.
14
15 Example 1:
16 Input: root = [3,9,20,null,null,15,7]
17 Output: [[9],[3,15],[20],[7]]
18 Explanation:
19 Column -1: Only node 9 is in this column.
20 Column 0: Nodes 3 and 15 are in this column in that order from top to bottom.
21 Column 1: Only node 20 is in this column.
22 Column 2: Only node 7 is in this column.
23
24 Example 2:
25 Input: root = [1,2,3,4,5,6,7]
26 Output: [[4],[2],[1,5,6],[3],[7]]
27 Explanation:
28 Column -2: Only node 4 is in this column.
29 Column -1: Only node 2 is in this column.
30 Column 0: Nodes 1, 5, and 6 are in this column.
31 |   |   | 1 is at the top, so it comes first.
32 |   |   | 5 and 6 are at the same position (2, 0), so we order them by their value, 5 before 6.
33 |   |   | Column 1: Only node 3 is in this column.
34 |   |   | Column 2: Only node 7 is in this column.
35
36 Example 3:
37 Input: root = [1,2,3,4,6,5,7]
38 Output: [[4],[2],[1,5,6],[3],[7]]
39 Explanation:
40 This case is the exact same as example 2, but with nodes 5 and 6 swapped.
41 Note that the solution remains the same since 5 and 6 are in the same location and should be ordered by their values.
42 */
```

Codeium: Refactor | Explain

```
51 class Node{
52     int data;
53     Node left;
54     Node right;
55     Node(int key){
56         this.data = key;
57     }
58 }
59
```

Codeium: Refactor | Explain

```
60 class Tuple{
61     Node node;
62     int row;
63     int col;
64     public Tuple(Node node, int row, int col){
65         this.node = node;
66         this.row = row;
67         this.col = col;
68     }
69 }
70
```

Codeium: Refactor | Explain

```
71 public class Vertical_Order_Traversal_of_Binary_Tree {
72     Codeium: Refactor | Explain | Generate Javadoc | X
73     public static List<List<Integer>> verticalTraversal(Node root) {
74         TreeMap<Integer, TreeMap<Integer, PriorityQueue<Integer>>> map = new TreeMap<>();
75         Queue<Tuple> q = new LinkedList<>();
76         q.add(new Tuple(root, 0, 0));
77         while(!q.isEmpty()){
78             Tuple tuple = q.remove();
79             Node node = tuple.node;
80             int x = tuple.row;
81             int y = tuple.col;
82
83             if(!map.containsKey(x)){
84                 map.put(x, new TreeMap<>());
85             }
86             if(!map.get(x).containsKey(y)){
87                 map.get(x).put(y, new PriorityQueue<>());
88             }
89             map.get(x).get(y).add(node.data);
90
91             if(node.left != null){
92                 q.add(new Tuple(node.left, x - 1, y + 1));
93             }
94             if(node.right != null){
95                 q.add(new Tuple(node.right, x + 1, y + 1));
96             }
97         }
98         List<List<Integer>> list = new ArrayList<>();
99         for(TreeMap<Integer, PriorityQueue<Integer>> ys : map.values()){
100             list.add(new ArrayList<>());
101             for(PriorityQueue<Integer> nodes : ys.values()){
102                 while(!nodes.isEmpty()){
103                     list.get(list.size() - 1).add(nodes.remove());
104                 }
105             }
106         }
107         return list;
108     }
109 }
```

Codeium: Refactor | Explain | Generate Javadoc | X

```

108  public static void main(String[] args) {
109      /*
110       Real World Representation the below tree into the code.
111       |
112       |   1
113       |  / \
114       | 2   3
115       | / \ / \
116       |4  5 6  7
117       | /   / \
118       |8   9   10
119      */
120      Node root = new Node(1);
121      root.left = new Node(2);
122      root.right = new Node(3);
123      root.left.left = new Node(4);
124      root.left.right = new Node(5);
125      root.left.right.left = new Node(8);
126      root.right.left = new Node(6);
127      root.right.right = new Node(7);
128      root.right.right.left = new Node(9);
129      root.right.right.right = new Node(10);
130      System.out.println(verticalTraversal(root));
131  }

```

> Task :Vertical_Order_Traversal_of_Binary_Tree.main()
[[4], [2, 8], [1, 5, 6], [3, 9], [7], [10]]

Binary Tree Zigzag Level Order Traversal

```

1  /*
2  103. Binary Tree Zigzag Level Order Traversal
3  Given the root of a binary tree, return the zigzag level order traversal of its nodes' values.
4  (i.e., from left to right, then right to left for the next level and alternate between).
5
6  Example 1:
7  Input: root = [3,9,20,null,null,15,7]
8  Output: [[3],[20,9],[15,7]]
9
10 Example 2:
11 Input: root = [1]
12 Output: [[1]]
13
14 Example 3:
15 Input: root = []
16 Output: []
17 */

```

```

33 public class Zig_Zag_Traversal_of_Binary_Tree {
34     public static List<List<Integer>> zigzagLevelOrder(Node root) {
35         List<List<Integer>> res = new ArrayList<>();
36         Queue<Node> q = new LinkedList<>();
37         if(root == null) return res;
38         q.add(root);
39         boolean flag = false;
40         while(!q.isEmpty()){
41             int k = q.size();
42             List<Integer> sublist = new ArrayList<>();
43             for(int i = 0; i < k; i++){
44                 if(q.peek().left != null) q.add(q.peek().left);
45                 if(q.peek().right != null) q.add(q.peek().right);
46                 if(flag){
47                     // Adding in reverse order by Adding at starting
48                     sublist.add(0, q.remove().data);
49                 }
50                 else{
51                     sublist.add(q.remove().data);
52                 }
53             }
54             flag = !flag;
55             res.add(sublist);
56         }
57         return res;
58     }
59 }
60
61 /*
62     Real World Representation the below tree into the code
63
64         1
65        / \
66       2   3
67      / \ / \
68     4  5 6  7
69    / \ / \
70   8  9 10
71
72 */
73 Node root = new Node(1);
74 root.left = new Node(2);
75 root.right = new Node(3);
76 root.left.left = new Node(4);
77 root.left.right = new Node(5);
78 root.left.right.left = new Node(8);
79 root.right.left = new Node(6);
80 root.right.right = new Node(7);
81 root.right.right.left = new Node(9);
82 root.right.right.right = new Node(10);
83 System.out.println(zigzagLevelOrder(root));

```

> Task :Zig_Zag_Traversal_of_Binary_Tree.main()
[[1], [3, 2], [4, 5, 6, 7], [10, 9, 8]]

