

# Trees TUF

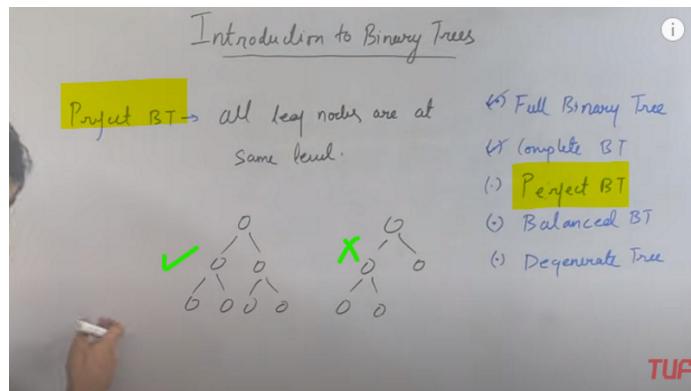
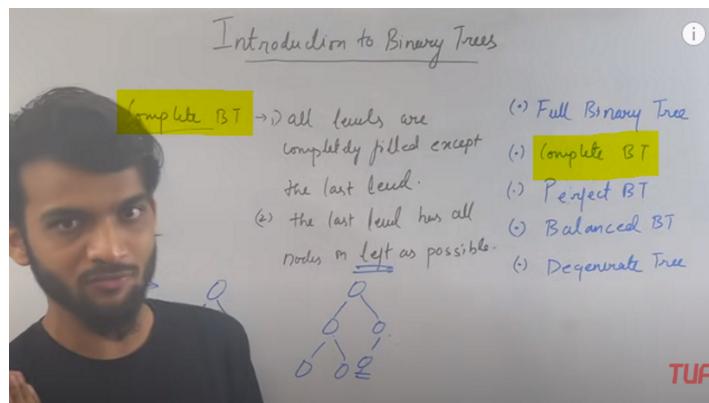
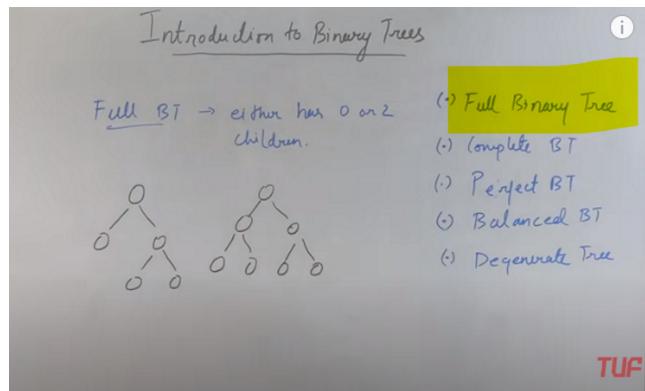
25 October 2021 18:24

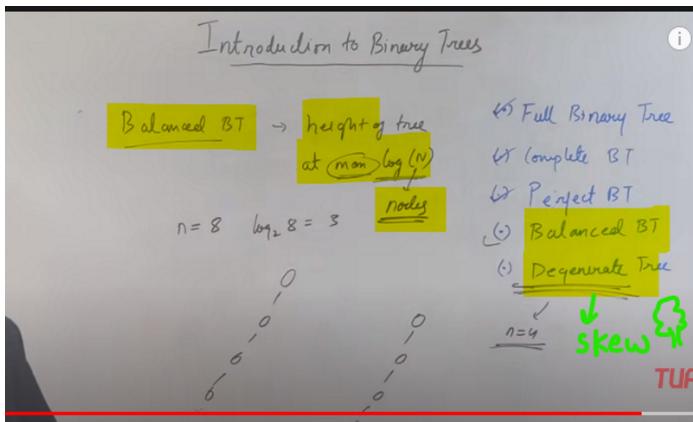
<https://techinterviewhandbook.org/algorithms/tree/>

<https://github.com/striver79/FreeKaTreeSeries>

\*\* => nice for understanding

Topic	Question	Difficulty	Link	Lec No	STATUS	Similar pblms
Trees						
	110	Easy	<a href="https://leetcode.com/problems/balanced-binary-tree/">https://leetcode.com/problems/balanced-binary-tree/</a>			
	543	Easy	<a href="https://leetcode.com/problems/diameter-of-binary-tree/">https://leetcode.com/problems/diameter-of-binary-tree/</a>			
	124 **	Hard(personally medium)	<a href="https://leetcode.com/problems/binary-tree-maximum-path-sum/">https://leetcode.com/problems/binary-tree-maximum-path-sum/</a>			
	987 **	Hard	<a href="https://leetcode.com/problems/vertical-order-traversal-of-a-binary-tree/">https://leetcode.com/problems/vertical-order-traversal-of-a-binary-tree/</a>	21		
	863**	med	<a href="https://leetcode.com/problems/all-nodes-distance-k-in-binary-tree/">https://leetcode.com/problems/all-nodes-distance-k-in-binary-tree/</a>			

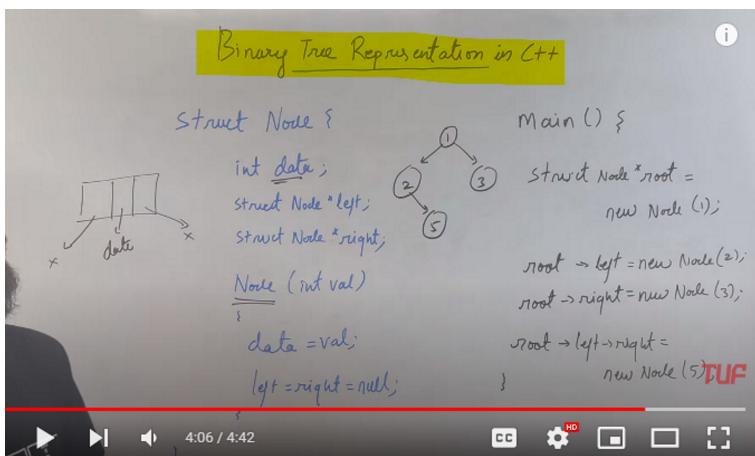
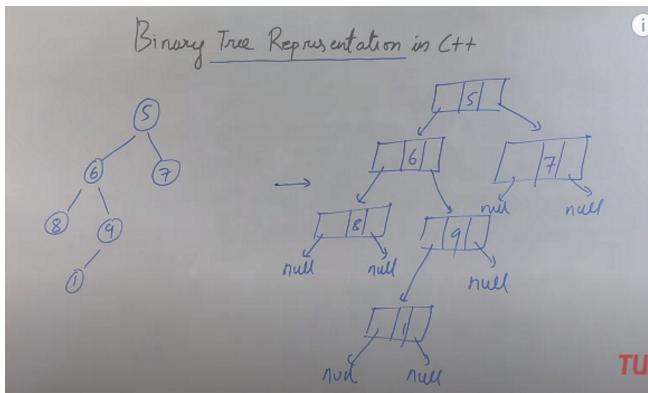




### Balanced binary tree

A balanced binary tree, also referred to as a height-balanced binary tree, is defined as a binary tree in which the height of the left and right subtree of any node differ by not more than 1. The difference between the left and the right subtree for any node is not more than one. The left subtree is balanced.

### L2. Binary Tree Representation in C++



### Another Representation of Binary Tree in C++ (as on leetcode):

```

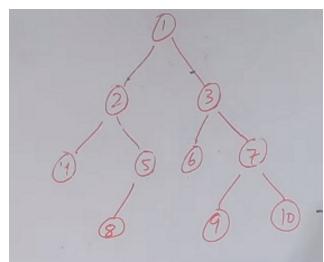
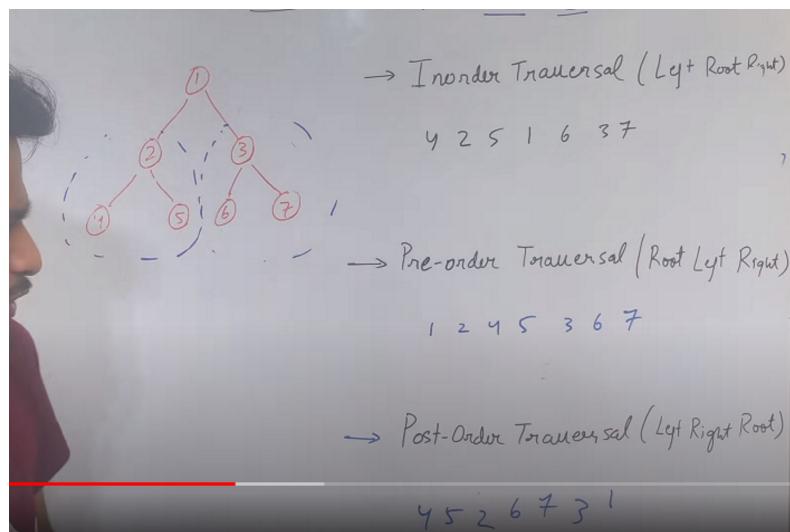
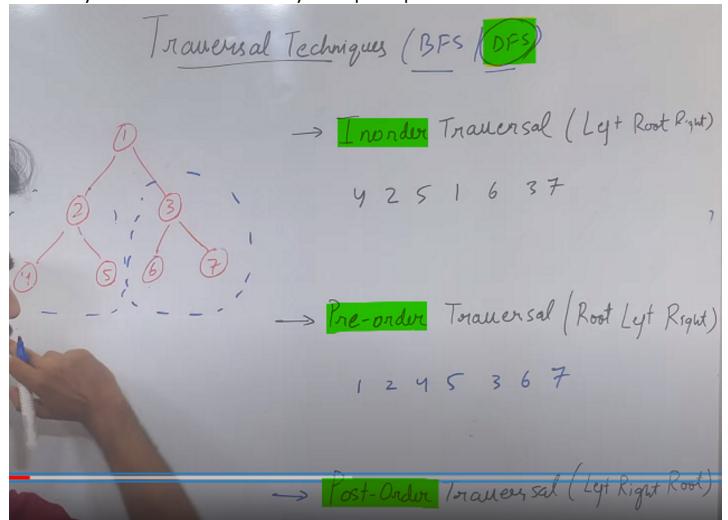
1 * 
2 * Definition for a binary tree node.
3 * struct TreeNode {
4 *     int val;
5 *     TreeNode *left;
6 *     TreeNode *right;
7 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
8 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
9 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
10 * };
11 */
12 class Solution {
13 public:
14     int maxDepth(TreeNode* root) {
    
```

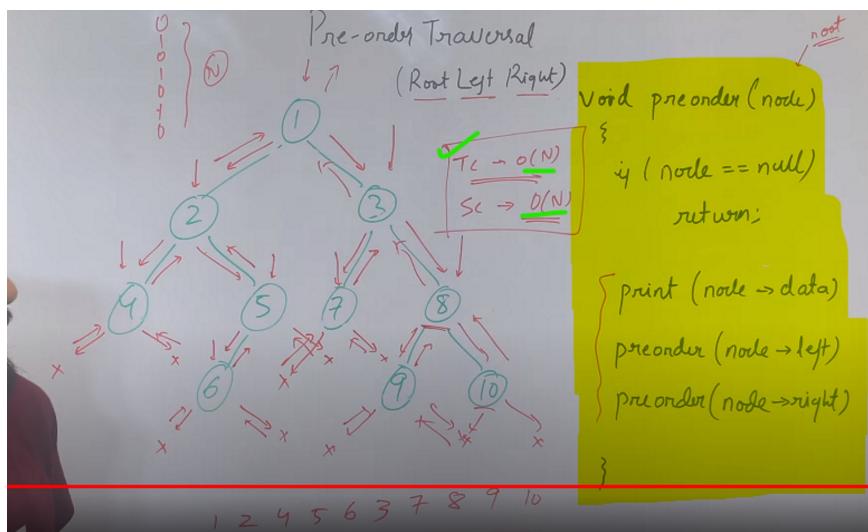
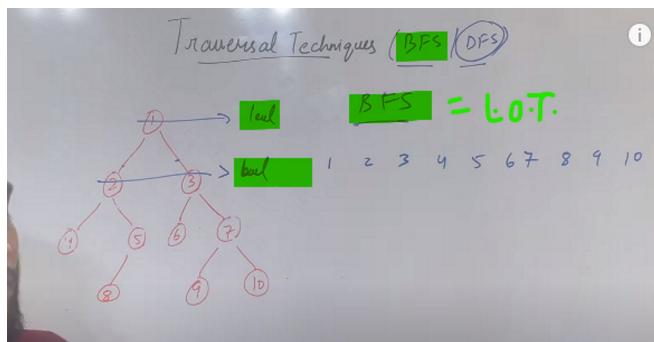
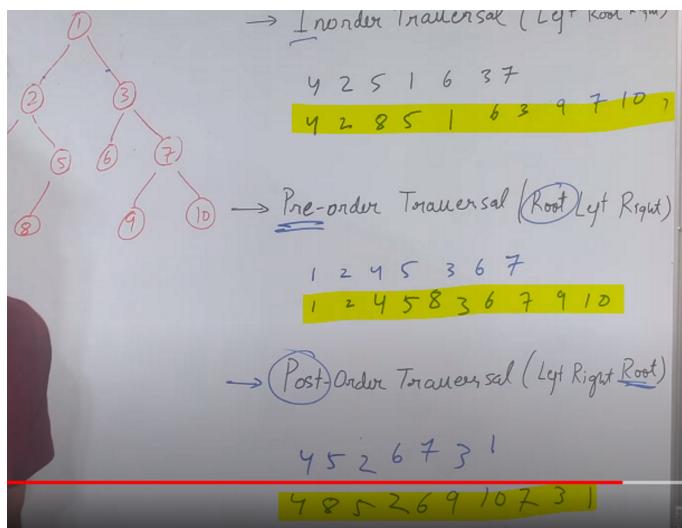
**\*\*Note\*\* :** Using `nullptr` is better than `null` for left and right tree pointers, as it prevents errors in many cases (and there is a logical explanation for that, because of data types)

Null => in c++ can mean both as a integer and as null reference ,and so it depends on the usage  
But nullptr : explicitly means that it is a pointer type, so its better to use it in case of pointer

=====

#### L4. Binary Tree Traversals in Binary Tree | BFS | DFS





// LC link : <https://leetcode.com/problems/binary-tree-preorder-traversal>

```

/**
 * Definition for a binary tree node.
 */
struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode() : val(0), left(nullptr), right(nullptr) {}
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
    TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
};

class Solution {
private:
    void dfs(TreeNode *node, vector<int> &preorder) {
        if(node == NULL) return;

```

```

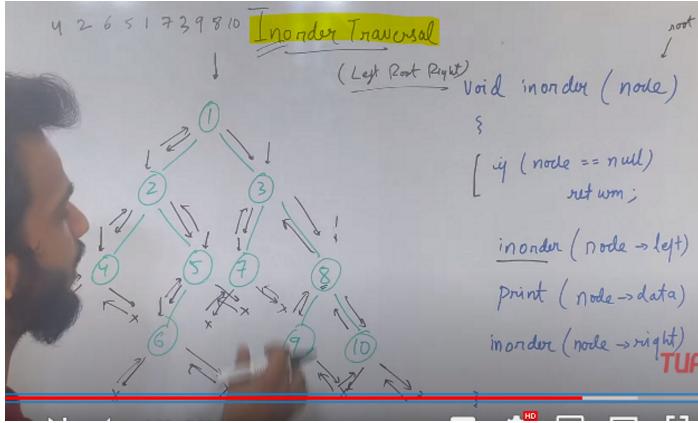
        preorder.push_back(node->val);
        dfs(node->left, preorder);
        dfs(node->right, preorder);
    }
public:
vector<int> preorderTraversal(TreeNode* root) {
    vector<int> preorder;
    dfs(root, preorder);
    return preorder;
}

```

---

- Q) Print no of nodes which have 0 children.  
 Q) Print no of nodes having just one subtree ,i.e, either only left subtree or only right subtree.
- 

#### L6. Inorder Traversal of Binary Tree



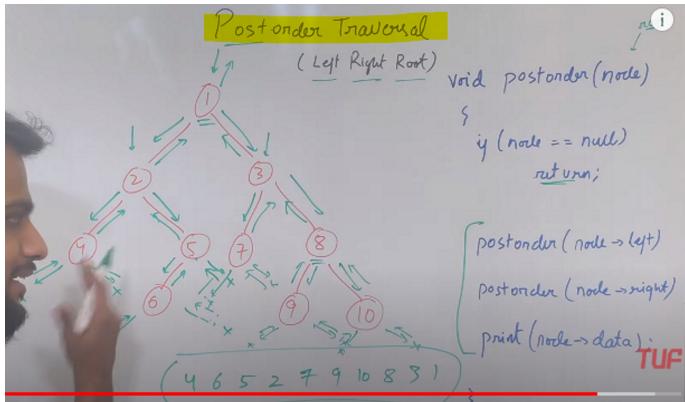
```

class Solution {
private:
void dfs(TreeNode *node, vector<int> &inorder) {
    if(node == NULL) return;

    dfs(node->left, inorder);
    inorder.push_back(node->val);
    dfs(node->right, inorder);
}
public:
vector<int> inorderTraversal(TreeNode* root) {
    vector<int> inorder;
    dfs(root, inorder);
    return inorder;
}

```

---



```

class Solution {
private:

```

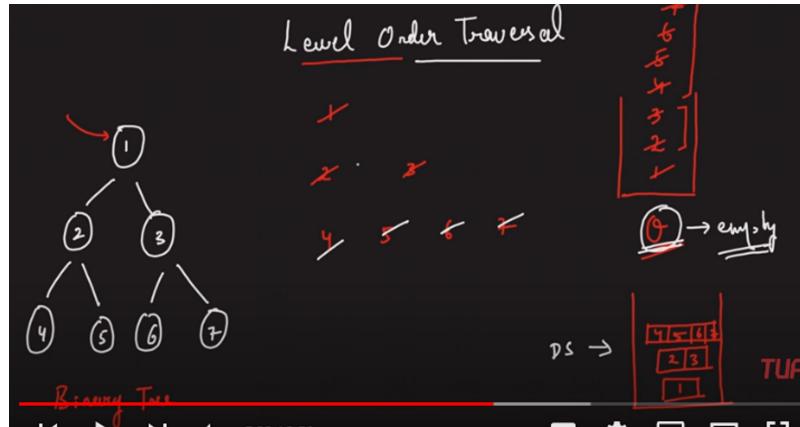
```

void dfs(TreeNode *node, vector<int> &postorder) {
    if(node == NULL) return;
    dfs(node->left, postorder);
    dfs(node->right, postorder);
    postorder.push_back(node->val);
}

public:
vector<int> postorderTraversal(TreeNode* root) {
    vector<int> postorder;
    dfs(root, postorder);
    return postorder;
}

```

From <https://github.com/striver79/FreeKaTreeSeries/blob/main/postorderCpp>



### L8. Level Order Traversal of Binary Tree | BFS | C++ | Java

```

/*
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 * };
 * TreeNode() : val(0), left(nullptr), right(nullptr) {}
 * TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 * TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 */
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        vector<vector<int>> ans;
        if(root == NULL) return ans;
        queue<TreeNode*> q;
        q.push(root);
        while(!q.empty()) {
            int size = q.size();
            vector<int> level;
            for(int i = 0;i<size;i++) {
                TreeNode *node = q.front();
                q.pop();
                if(node->left != NULL) q.push(node->left);
                if(node->right != NULL) q.push(node->right);
                level.push_back(node->val);
            }
            ans.push_back(level);
        }
        return ans;
    }
};


```

Previous code was restored from your local storage. Reset to default

```

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<TreeNode>();
        List<List<Integer>> wrapList = new LinkedList<List<Integer>>();
        if(root == null) return wrapList;
        queue.offer(root);
        while(!queue.isEmpty()){
            int levelNum = queue.size();
            List<Integer> sublist = new LinkedList<Integer>();
            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;
    }
}


```

Your previous code was restored from your local storage. Reset to default

=====

**L9. Iterative Preorder Traversal**  
TC= O(n), SC = O(n) worst case tree shown in green below)

TUF

L9. Iterative Preorder Traversal in Binary Tree | C++ | Java | Stack

```

1  /**
2  * Definition for a binary tree node.
3  * struct TreeNode {
4  *     int val;
5  *     TreeNode *left;
6  *     TreeNode *right;
7  *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
8  *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
9  *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
10 }
11 */
12 class Solution {
13
14 public:
15     vector<int> preorderTraversal(TreeNode* root) {
16         vector<int> preorder;
17         if(root == NULL) return preorder;
18
19         stack<TreeNode*> st;
20         st.push(root);
21         while(!st.empty()){
22             root = st.top();
23             st.pop();
24             preorder.push_back(root->val);
25             if(root->right != NULL){
26                 st.push(root->right);
27             }
28             if(root->left!= NULL){
29                 st.push(root->left);
30             }
31         }
32         return preorder;
33     }
34 };

```

```

1 // LC: https://leetcode.com/problems/binary-tree-preorder-traversal
2 /**
3  * Definition for a binary tree node.
4  * public class TreeNode {
5  *     int val;
6  *     TreeNode left;
7  *     TreeNode right;
8  *     TreeNode() {}
9  *     TreeNode(int val) { this.val = val; }
10    TreeNode(int val, TreeNode left, TreeNode right) {
11        this.val = val;
12        this.left = left;
13        this.right = right;
14    }
15 }
16 */
17 class Solution {
18
19     public List<Integer> preorderTraversal(TreeNode root) {
20         List<Integer> preorder = new ArrayList<Integer>();
21         if(root == null) return preorder;
22         Stack<TreeNode> st = new Stack<TreeNode>();
23         st.push(root);
24         while(!st.isEmpty()){
25             root = st.pop();
26             preorder.add(root.val);
27             if(root.right != null){
28                 st.push(root.right);
29             }
30             if(root.left!= null){
31                 st.push(root.left);
32             }
33         }
34         return preorder;
35     }

```

TC → O(N)  
SC → O(N)

Your previous code was restored from your local storage. Reset to default

```

1 /**
2  * Definition for a binary tree node.
3  * struct TreeNode {
4  *     int val;
5  *     TreeNode *left;
6  *     TreeNode *right;
7  *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
8  *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
9  *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
10 }
11 */
12 class Solution {
13
14 public:
15     //approach 1: recursive
16     //approach 2: iterative (current)
17     vector<int> preorderTraversal(TreeNode* root) {
18         vector<int> ans;
19         if(root==nullptr) return ans;
20
21         stack<TreeNode*> s;
22         s.push(root);
23         while(!s.empty()){
24             ans.push_back(s.top()->val);
25             TreeNode* temp=s.top();
26             s.pop();
27             if(temp->right!=nullptr) s.push(temp->right);
28             if(temp->left!=nullptr) s.push(temp->left);
29         }
30         return ans;
31     }

```

**Note:** Here for preOrder traversal we push the right child first and then the left child in the stack first as we want to print left node first and that is only possible if right child is pushed first followed by left so that left remains on the top!!

---

**L10 : Iterative Inorder Traversal**  
**TC= O(n) , SC= O(n) (for left skewed tree)**

```

1 class Solution {
2
3     public:
4         vector<int> inorderTraversal(TreeNode* root) {
5             stack<TreeNode*> st;
6             TreeNode* node = root;
7             vector<int> inorder;
8             while(true) {
9                 if(node != NULL) {
10                     st.push(node);
11                     node = node->left;
12                 }
13                 else {
14
15                     if(st.empty() == true) break;
16                     node = st.top();
17                     st.pop();
18                     inorder.push_back(node->val);
19                     node = node->right;
20                 }
21             }
22             return inorder;
23         }
24     };

```

=====

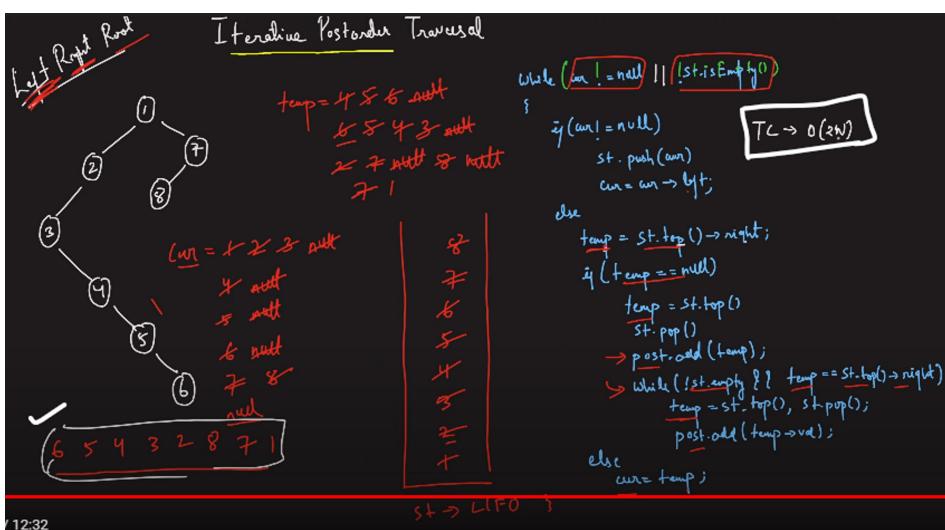
**L11: Iterative Postorder traversal with 2 stacks**

TC=O(N) , SC= O(N)

=====

**L12: Iterative Postorder traversal with single stack**

TC=O(2\*N) ,Auxillary SC= O(N) (used by stack)



**L13: Pre + In + Post : Order ,All 3 traversals in just one go(/traversal)**

- TC= O(3\*N) -> since each node is visited 3 times
- Auxiliary SC= O(N) : Used for stack of pair,
- Total SC= O(4\*N)
  - 3\*n for 3 lists :in,pre,post
  - n for stack of pair

L13. Preorder Inorder Postorder Traversals in One Traversal | C++ | Java | Stack | Binary Trees

Preorder / Inorder / Postorder

Preorder → 1 2 3 4 5 6 7  
Inorder → 3 2 4 1 6 5 7  
Postorder → 3 4 2 6 7 5 1

TC →  $O(3 \times N)$   
SC →  $O(4N)$

Stack: (node, num)  
LIFO  
if num == 3  
TUF

C++

```

13 */
14 class Solution {
15 public:
16     vector<int> preInPostTraversal(TreeNode* root) {
17         stack<pair<TreeNode*, int>> st;
18         st.push({root, 1});
19         vector<int> pre, in, post;
20         if(root == NULL) return {};
21         while(st.empty() == false) {
22             auto it = st.top();
23             st.pop();
24
25             // this is part of pre
26             // increment 1 to 2
27             // push the left side of the tree
28             if(it.second == 1) {
29                 pre.push_back(it.first->val);
30                 it.second++;
31                 st.push(it);
32
33                 if(it.first->left != NULL) {
34                     st.push({it.first->left, 1});
35                 }
36             }
37
38             // this is a part of in
39             // increment 2 to 3
40             // push right
41             else if(it.second == 2) {
42                 in.push_back(it.first->val);
43                 it.second++;
44                 st.push(it);
45
46                 if(it.first->right != NULL) {
47                     st.push({it.first->right, 1});
48                 }
49             }
50             // don't push it back again
51             else {
52                 post.push_back(it.first->val);
53             }
54         }
55     }
56 };

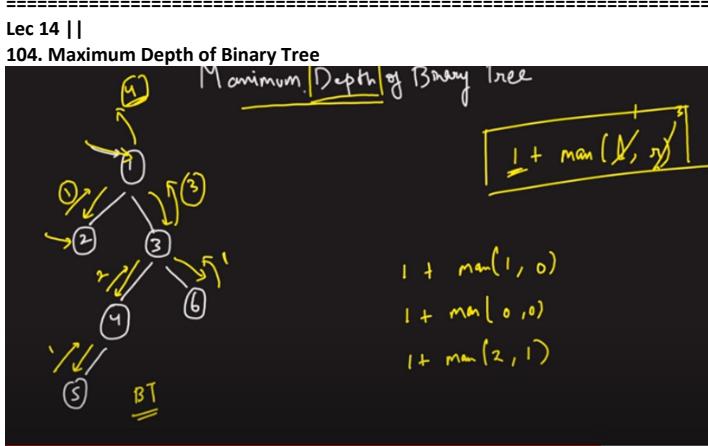
```

Java

```

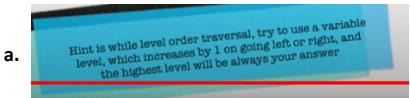
23 }
24 class Solution {
25     public void preInPostTraversal(TreeNode root) {
26         Stack<Pair> st = new Stack<Pair>();
27         st.push(new Pair(root, 1));
28         List<Integer> pre = new ArrayList<>();
29         List<Integer> in = new ArrayList<>();
30         List<Integer> post = new ArrayList<>();
31         if(root == null) return;
32
33         while(st.isEmpty() == false) {
34             Pair it = st.pop();
35
36             // this is part of pre
37             // increment 1 to 2
38             // push the left side of the tree
39             if(it.num == 1) {
40                 pre.add(it.node.val);
41                 it.num++;
42                 st.push(it);
43
44                 if(it.node.left != null) {
45                     st.push(new Pair(it.node.left, 1));
46                 }
47             }
48
49             // this is a part of in
50             // increment 2 to 3
51             // push right
52             else if(it.num == 2) {
53                 in.add(it.node.val);
54                 it.num++;
55                 st.push(it);
56
57                 if(it.node.right != null) {
58                     st.push(new Pair(it.node.right, 1));
59                 }
60             }
61             // don't push it back again
62             else {
63                 post.add(it.node.val);
64             }
65         }
66     }
67 }

```



There are 2 ways to find the max depth of binary tree:

1. Using recursion
2. Using level order traversal



### Soln 1: Recursive:

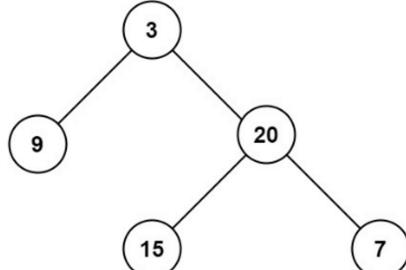
#### 104. Maximum Depth of Binary Tree

Easy 5074 106 Add to List Share

Given the `root` of a binary tree, return its *maximum depth*.

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.

#### Example 1:



Input: `root = [3,9,20,null,null,15,7]`

```

1 /**
2  * Definition for a binary tree node.
3  * struct TreeNode {
4  *     int val;
5  *     TreeNode *left;
6  *     TreeNode *right;
7  *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
8  *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
9  *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
10 * };
11 */
12 class Solution {
13 public:
14     int maxDepth(TreeNode* root) {
15         if(root==nullptr) return 0;
16         return 1+max(maxDepth(root->left),maxDepth(root->right));
17     }
18 };
  
```

Testcase Run Code Result Debugger

Accepted Runtime: 0 ms

Your input `[3,9,20,null,null,15,7]`

Output `3`

Expected `3`

### Soln2: Level order traversal

LeetCode Explore Problems Interview Contest Discuss Store

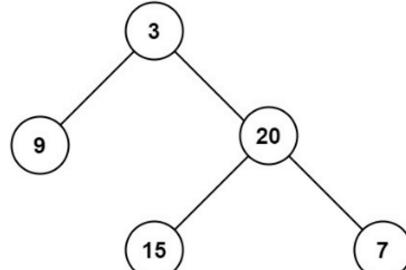
#### 104. Maximum Depth of Binary Tree

Easy 5074 106 Add to List Share

Given the `root` of a binary tree, return its *maximum depth*.

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.

#### Example 1:



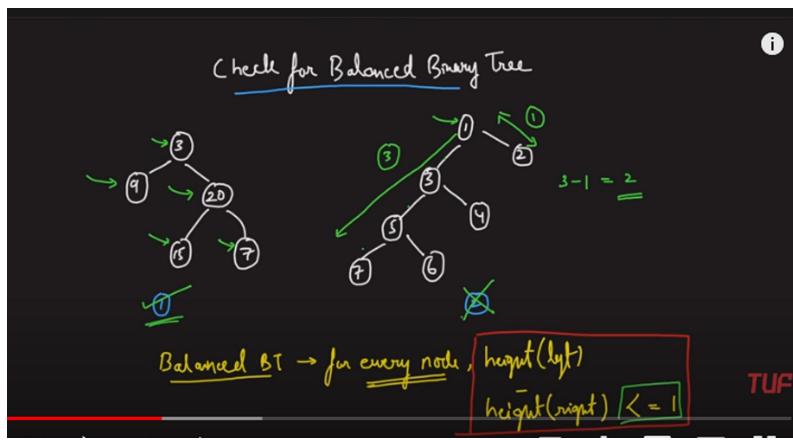
Input: `root = [3,9,20,null,null,15,7]`

### L15. Check for Balanced Binary Tree | C++ | Java

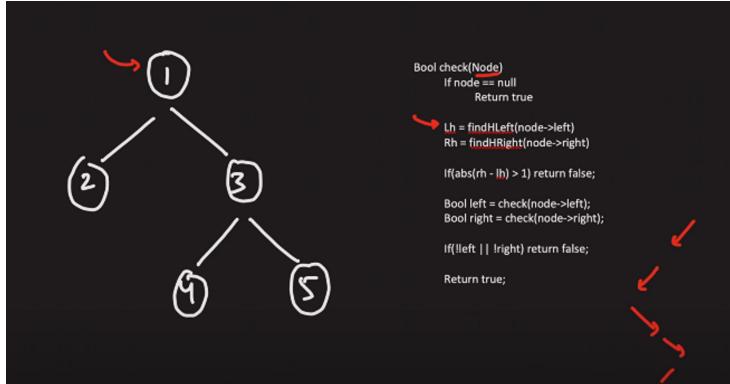
<https://leetcode.com/problems/balanced-binary-tree/>

```

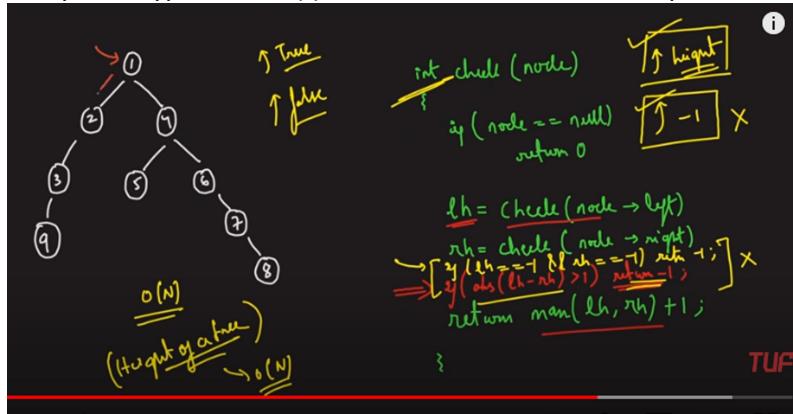
i C++ v Autocomplete
1 /**
2  * Definition for a binary tree node.
3  * struct TreeNode {
4  *     int val;
5  *     TreeNode *left;
6  *     TreeNode *right;
7  *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
8  *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
9  *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
10 * };
11 */
12 class Solution {
13 public:
14     //approach 1: recursion (this is better)
15     //approach 2: by level order traversal [current]
16     int maxDepth(TreeNode* root) {
17         //level order traversal
18         int level=0;
19         if(root==nullptr) return 0;
20         queue<TreeNode*> q;
21         q.push(root);
22         while(!q.empty()){
23             int n=q.size();
24             for(int i=0;i<n;i++){
25                 if(q.front()->left != nullptr) q.push(q.front()->left);
26                 if(q.front()->right != nullptr) q.push(q.front()->right);
27                 q.pop();
28             }
29             level++;
30         }
31         return level;
32     }
33 };
  
```



### 1. Brute Force Approach : TC= O(n\*n)



### 2. Optimized Approach : TC = O(n) $\rightarrow$ we make modifications to the maxDepth of a tree code



### O(n) solution of striver:

```

1 * Definition for a binary tree node.
2 * struct TreeNode {
3 *     int val;
4 *     TreeNode *left;
5 *     TreeNode *right;
6 * };
7 * TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
8 * TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
9 *
10 */
11
12
13 class Solution {
14 public:
15     bool isBalanced(TreeNode* root) {
16         return dfsHeight (root) != -1;
17     }
18     int dfsHeight (TreeNode* root) {
19         if (root == NULL) return 0;
20
21         int leftHeight = dfsHeight (root->left);
22         if (leftHeight == -1) return -1;
23         int rightHeight = dfsHeight (root->right);
24         if (rightHeight == -1) return -1;
25
26         if (abs(leftHeight - rightHeight) > 1) return -1;
27         return max (leftHeight, rightHeight) + 1;
28     }
29 };
    
```

```

1 * Definition for a binary tree node.
2 * public class TreeNode {
3 *     int val;
4 *     TreeNode left;
5 *     TreeNode right;
6 * };
7 * TreeNode(int val) { this.val = val; }
8 * TreeNode(int val, TreeNode left, TreeNode right) {
9 *     this.val = val;
10 *     this.left = left;
11 *     this.right = right;
12 * }
13 *
14 */
15
16 class Solution {
17     public boolean isBalanced(TreeNode root) {
18         return dfsHeight (root) != -1;
19     }
20     int dfsHeight (TreeNode root) {
21         if (root == null) return 0;
22
23         int leftHeight = dfsHeight (root.left);
24         if (leftHeight == -1) return -1;
25         int rightHeight = dfsHeight (root.right);
26         if (rightHeight == -1) return -1;
27
28         if (Math.abs(leftHeight - rightHeight) > 1) return -1;
29         return Math.max(leftHeight, rightHeight) + 1;
30     }
31 }
    
```

### My O(n) solution(same concept as striver but a bit different implementation):

LeetCode Explore Problems Interview Contest Discuss Store

October LeetCoding Challenge 2021 Premium

Description Solution Discuss (999+) Submissions

**110. Balanced Binary Tree**

Easy 4505 258 Add to List Share

Given a binary tree, determine if it is height-balanced.

For this problem, a height-balanced binary tree is defined as:

- a binary tree in which the left and right subtrees of every node differ in height by no more than 1.

**Example 1:**

```

class Solution {
public:
    int maxDepth(TreeNode* node, bool &ans) {
        if(node==NULL || ans==false) return 0;
        int lh=maxDepth(node->left,ans);
        int rh=maxDepth(node->right,ans);
        if(abs(lh-rh)>=2){
            ans=false;
        }
        return 1+max(lh,rh);
    }
    bool isBalanced(TreeNode* root) {
        bool ans=true;
        int d=maxDepth(root,ans);
        return ans;
    }
};

```

Testcase Run Code Result Debugger

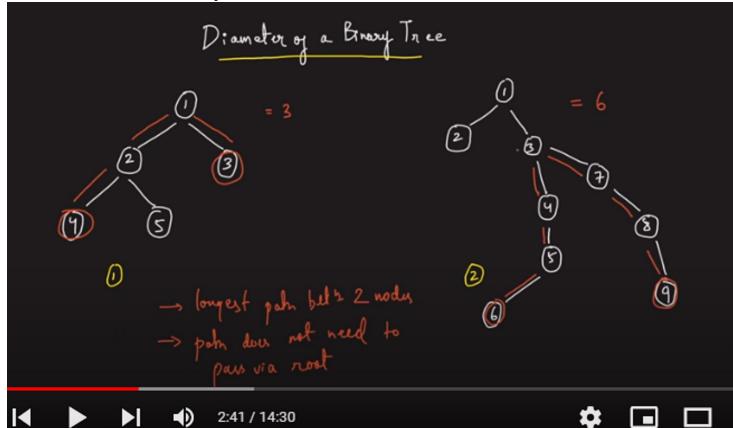
Accepted Runtime: 0 ms

Your input [1,2,2,3,3,null,null,4,4]

Output false

Expected false

### L16.Diameter of a Binary Tree

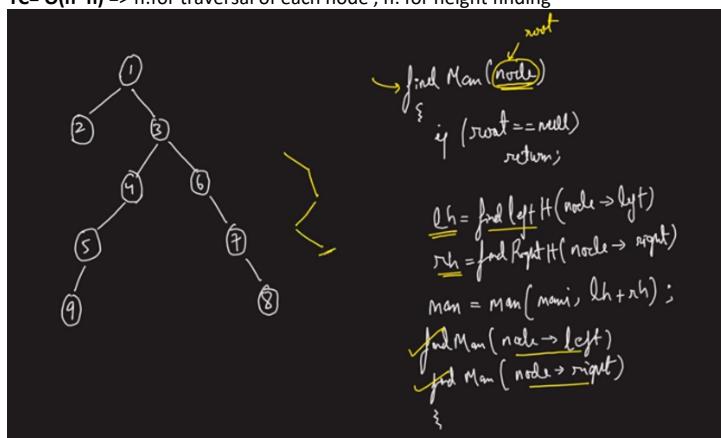


### Diameter of a binary tree:

- Longest path between any 2 nodes (path = no of edges bw any 2 nodes )
- Path need not pass via root

**Brute-force Approach :** Find left and right height of tree for each node and  $\text{ans} = \max(\text{left\_height} + \text{right\_height}, \text{ans})$ , for each node

TC=  $O(n^2)$  => n:for traversal of each node , n:for height finding

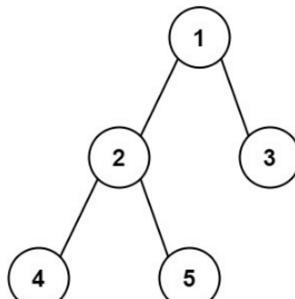


## 543. Diameter of Binary Tree

Easy 6272 388 Add to List Share

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:



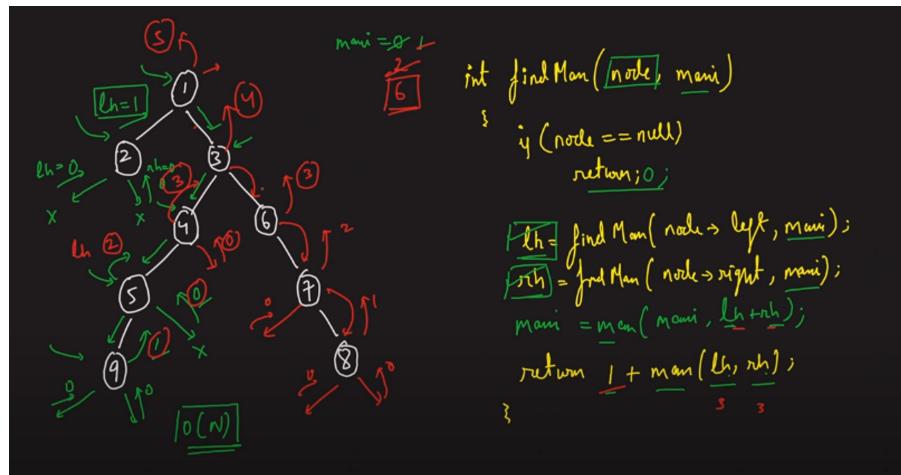
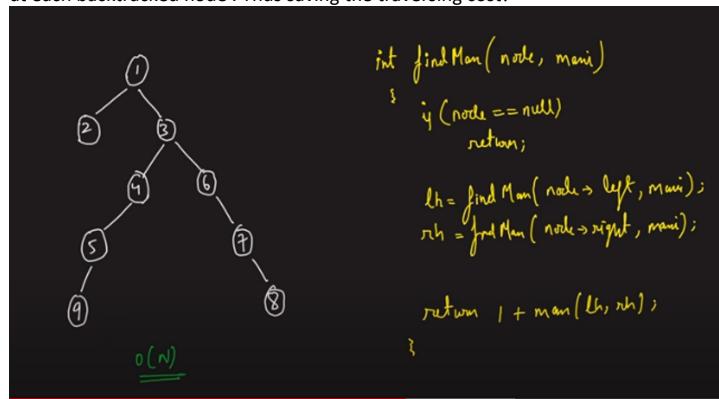
```

1 * /**
2 * Definition for a binary tree node.
3 * struct TreeNode {
4 *     int val;
5 *     TreeNode *left;
6 *     TreeNode *right;
7 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
8 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
9 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
10 * };
11 */
12 class Solution {
13 public:
14     int maxDepth(TreeNode* node){
15         if(node==nullptr) return 0;
16         return 1+max(maxDepth(node->left),maxDepth(node->right));
17     }
18
19 void traverse(TreeNode* node,int &ans){
20     if(node==nullptr) return;
21
22     int lh=maxDepth(node->left);
23     int rh=maxDepth(node->right);
24     ans=max(ans,lh+rh);
25     traverse(node->left,ans);
26     traverse(node->right,ans);
27 }
28
29 int diameterOfBinaryTree(TreeNode* root) {
30     int ans=0,lh,rh;
31     if(root==nullptr) return ans;
32
33     traverse(root,ans);
34     return ans;
35 }
36

```

## Optimized Approach :

The only difference here is that we don't traverse separately to each node, rather use the backtracking to achieve the same and update the diameter(which is passed by reference) at each backtracked node . Thus saving the traversing cost!



★  $Tc \rightarrow O(N)$   
 $Sc \rightarrow O(N)$

```

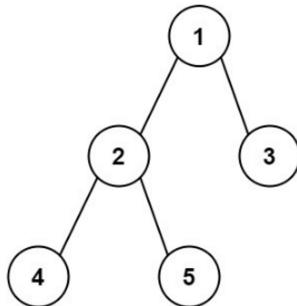
1+ /**
2+ * Definition for a binary tree node.
3+ * struct TreeNode {
4+ *     int val;
5+ *     TreeNode *left;
6+ *     TreeNode *right;
7+ *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
8+ *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
9+ *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
10+ * };
11+ */
12+
13+
14+
15+
16+
17+ class Solution {
18+ public:
19+     int diameterOfBinaryTree(TreeNode* root) {
20+         int diameter = 0;
21+         height(root, diameter);
22+         return diameter;
23+     }
24+ private:
25+     int height(TreeNode* node, int& diameter) {
26+         if (node == nullptr) {
27+             return 0;
28+         }
29+         int lh = height(node->left, diameter);
30+         int rh = height(node->right, diameter);
31+         diameter = max(diameter, lh + rh);
32+         return 1 + max(lh, rh);
33+     }
34+ };

```

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

## 543. Diameter

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

```

12 //approach 2: optimised
13 class Solution {
14 public:
15     int diameterOfBinaryTree(TreeNode* root) {
16         int diameter=0;
17         if(root==nullptr) return 0;
18         height(root,diameter); // int x= ....
19         return diameter;
20     }
21     int height(TreeNode* node,int &diameter){
22         if(node==nullptr) return 0;
23
24         int lh=height(node->left,diameter);
25         int rh=height(node->right,diameter);
26         diameter=max(diameter,lh+rh);
27
28         return 1+max(lh,rh);
29     }
30 }
31 /*
32 // approach 1: brute force
33 class Solution {
34 public:

```

Testcase Run Code Result Debugger

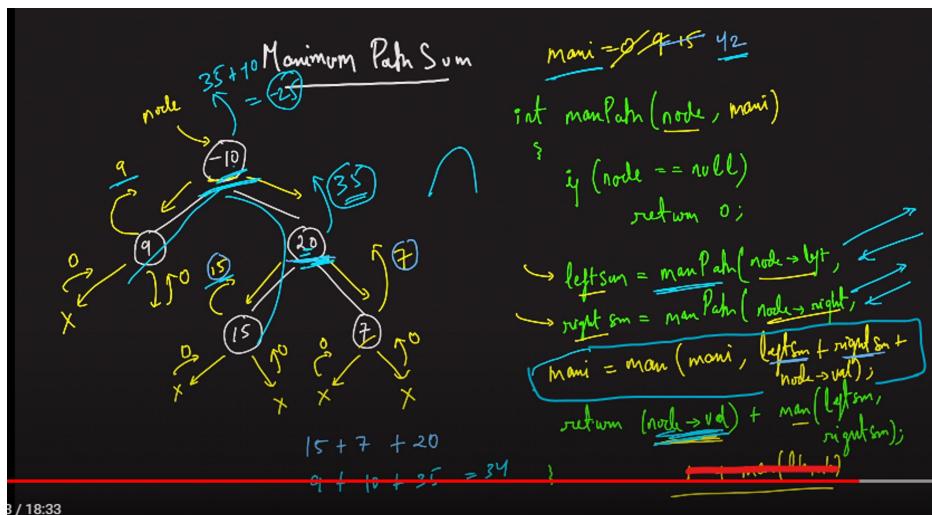
Accepted Runtime: 0 ms

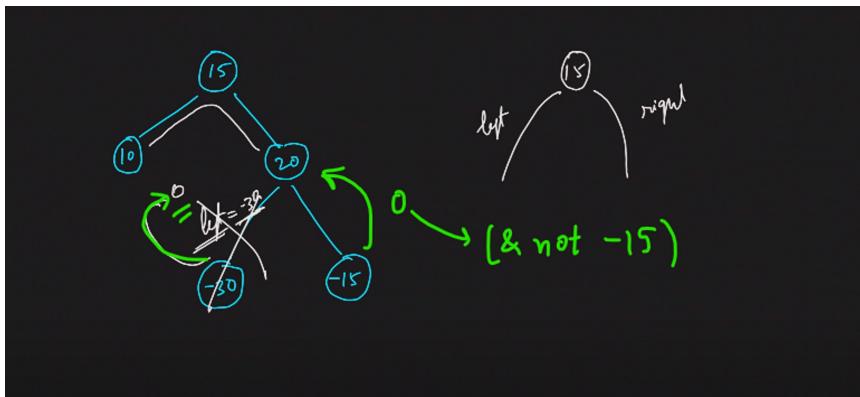
Your input [1,2,3,4,5]

Output 3

Expected 3

## L17. 124 Binary Tree Max Path Sum





If left/right path sum is negative then return 0 !

Striver's Solution:

<pre> 1 C++    * Autocomplete 2 /** 3  * Definition for a binary tree node. 4  * struct TreeNode { 5  *     int val; 6  *     TreeNode *left; 7  *     TreeNode *right; 8  *     TreeNode(): val(0), left(nullptr), right(nullptr) {} 9  *     TreeNode(int x): val(x), left(nullptr), right(nullptr) {} 10 *     TreeNode(int x, TreeNode *left, TreeNode *right): val(x), left(left), right(right) {} 11 */ 12 class Solution { 13 public: 14     int maxPathSum(TreeNode* root) { 15         int maxi = INT_MIN; 16         maxPathDown(root, maxi); 17         return maxi; 18     } 19 20     int maxPathDown(TreeNode* node, int &amp;maxi) { 21         if (node == NULL) return 0; 22         int left = max(0, maxPathDown(node-&gt;left, maxi)); 23         int right = max(0, maxPathDown(node-&gt;right, maxi)); 24         maxi = max(maxi, left + right + node-&gt;val); 25         return max(left, right) + node-&gt;val; 26     } 27 }; </pre>	<pre> 1 Java    * Autocomplete 2 /** 3  * Definition for a binary tree node. 4  * public class TreeNode { 5  *     int val; 6  *     TreeNode left; 7  *     TreeNode right; 8  *     TreeNode() {} 9  *     TreeNode(int val) { this.val = val; } 10 *     TreeNode(int val, TreeNode left, TreeNode right) { 11 *         this.val = val; 12 *         this.left = left; 13 *         this.right = right; 14 *     } 15 */ 16 class Solution { 17     public int maxPathSum(TreeNode root) { 18         int maxValue[] = new int[1]; 19         maxValue[0] = Integer.MIN_VALUE; 20         maxPathDown(root, maxValue); 21         return maxValue[0]; 22     } 23 24     private int maxPathDown(TreeNode node, int maxValue[]) { 25         if (node == null) return 0; 26         int left = Math.max(0, maxPathDown(node.left, maxValue)); 27         int right = Math.max(0, maxPathDown(node.right, maxValue)); 28         maxValue[0] = Math.max(maxValue[0], left + right + node.val); 29         return Math.max(left, right) + node.val; 30     } 31 } </pre>
---	--

My Solution:

Description Solution Discuss (999+) Submissions C++ Autocomplete

**124. Binary Tree Maximum Path Sum**

Hard 7592 461 Add to List Share

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. A node can only appear in the sequence at most once. Note that the path does not need to pass through the root.

The path sum of a path is the sum of the node's values in the path.

Given the root of a binary tree, return the maximum path sum of any non-empty path.

**Example 1:**

```

    graph TD
      1((1)) --> 2((2))
      1 --> 3((3))
  
```

Input: root = [1,2,3]  
 Output: 6  
 Explanation: The optimal path is 2 -> 1 -> 3 with a path sum of 2 + 1 + 3 =

```

5 *   TreeNode *left;
6 *   TreeNode *right;
7 *   TreeNode() : val(0), left(nullptr), right(nullptr) {}
8 *   TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
9 *   TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
10 *
11 */
12 class Solution {
13 public:
14     int maxPathSum(TreeNode* root) {
15         // if(root->)
16         int sum=0,ans=INT_MIN,i;
17         sum=pathSum(root,ans);
18         // ans= ans<0 ? 0 : ans;
19         return ans;
20     }
21     int pathSum(TreeNode* node,int& ans){
22         if(node==nullptr) return 0;
23
24         int maxLeft = pathSum(node->left,ans);
25         int maxRight = pathSum(node->right,ans);
26         //sum is the maximum path sum from current node and either its left subtree / its right subtree
27         //note its a straightline sum for its ancestor node's v-path sum
28         int sum = max(maxRight+node->val,maxLeft+node->val);
29         sum = max(sum,node->val);
30
31         //ans checks & updates: if the current node's v-path sum is the max path sum
32         ans = max(ans, node->val + maxLeft + maxRight);
33         ans = max(ans, node->val + maxLeft, node->val + maxRight);
34         ans = max(ans, node->val);
35
36         return sum;
37     }
38 };
  
```

Controls: Download Open Editor Next i Dm Code Submit

### L18. Same Tree or not

C++ Autocomplete L18. Check it two trees are Identical or Not | C++ | Java

```

1 /**
2  * Definition for a binary tree node.
3  * struct TreeNode {
4  *     int val;
5  *     TreeNode *left;
6  *     TreeNode *right;
7  *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
8  *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
9  *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
10 */
11
12
13
14
15
16
17 class Solution {
18 public:
19     bool isSameTree(TreeNode* p, TreeNode* q) {
20         if(p==NULL || q == NULL) {
21             return (p==q);
22         }
23         return (p->val == q->val)
24             && isSameTree(p->left, q->left)
25             && isSameTree(p->right, q->right);
26     }
27 };
  
```

TC → O(N)  
 SC → O(N)

Java Autocomplete

```

1 /**
2  * Definition for a binary tree node.
3  * public class TreeNode {
4  *     int val;
5  *     TreeNode left;
6  *     TreeNode right;
7  *     TreeNode() {}
8  *     TreeNode(int val) { this.val = val; }
9  *     TreeNode(int val, TreeNode left, TreeNode right) {
10        this.val = val;
11        this.left = left;
12        this.right = right;
13    }
14 }
15
16 class Solution {
17     public boolean isSameTree(TreeNode p, TreeNode q) {
18         if(p==null || q == null) {
19             return (p==q);
20         }
21         return (p.val == q.val) && isSameTree(p.left, q.left) && isSameTree(p.right, q.right);
22     }
23 }
  
```

Description Solution Discuss (999+) Submissions

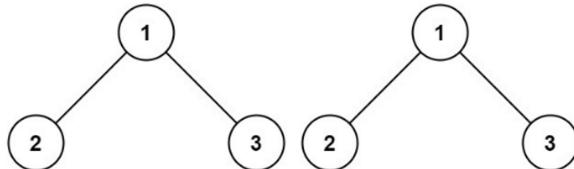
## 100. Same Tree

Easy 4319 110 Add to List Share

Given the roots of two binary trees  $p$  and  $q$ , write a function to check if they are the same or not.

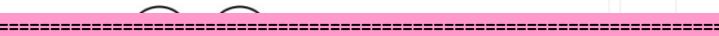
Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.

## Example 1:



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

## Example 2:

L19.Zig-Zag or Spiral Traversal in binary tree

## Striver Solution:

TC= O(N) , SC = O(N)

```

1 * Autocomplete
2 * Definition for a binary tree node.
3 * struct TreeNode {
4 *     int val;
5 *     TreeNode *left;
6 *     TreeNode *right;
7 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
8 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
9 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
10 */
11
12 class Solution {
13 public:
14     vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
15         vector<vector<int>> result;
16         if (root == NULL) {
17             return result;
18         }
19
20         queue<TreeNode*> nodesQueue;
21         nodesQueue.push(root);
22         bool leftToRight = true;
23
24         while (!nodesQueue.empty()) {
25             int size = nodesQueue.size();
26             vector<int> row(size);
27             for (int i = 0; i < size; i++) {
28                 TreeNode* node = nodesQueue.front();
29                 nodesQueue.pop();
30
31                 // find position to fill node's value
32                 int index = (leftToRight) ? i : (size - 1 - i);
33
34                 row[index] = node->val;
35                 if (node->left) {
36                     nodesQueue.push(node->left);
37                 }
38                 if (node->right) {
39                     nodesQueue.push(node->right);
40                 }
41             }
42             // after this level
43             leftToRight = !leftToRight;
44             result.push_back(row);
45         }
46     }
47 }
  
```

```

1 * Autocomplete
2 * Definition for a binary tree node.
3 * struct TreeNode {
4 *     int val;
5 *     TreeNode *left;
6 *     TreeNode *right;
7 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
8 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
9 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
10 */
11
12 class Solution {
13 public:
14     vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
15         vector<vector<int>> result;
16         if (root == NULL) {
17             return result;
18         }
19
20         queue<TreeNode*> nodesQueue;
21         nodesQueue.push(root);
22         bool leftToRight = true;
23
24         while (!nodesQueue.empty()) {
25             int size = nodesQueue.size();
26             vector<int> row(size);
27             for (int i = 0; i < size; i++) {
28                 TreeNode* node = nodesQueue.front();
29                 nodesQueue.pop();
30
31                 // find position to fill node's value
32                 int index = (leftToRight) ? i : (size - 1 - i);
33
34                 row[index] = node->val;
35                 if (node->left) {
36                     nodesQueue.push(node->left);
37                 }
38                 if (node->right) {
39                     nodesQueue.push(node->right);
40                 }
41             }
42             // after this level
43             leftToRight = !leftToRight;
44             result.push_back(row);
45         }
46     }
47 }
  
```

## My Solution:

TC,SC : Same as striver's (just that striver did in-place reversal &amp; I ran a loop)

Console Contribute 5:44 / 9:02

Run Code

Console Contribute

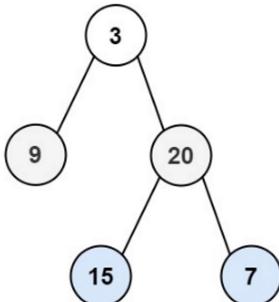
HD Run C

### 103. Binary Tree Zigzag Level Order Traversal

Medium 4563 147 Add to List Share

Given the root of a binary tree, return the zigzag level order traversal of its nodes' values. (i.e., from left to right, then right to left for the next level and alternate between).

#### Example 1:



Input: root = [3,9,20,null,null,15,7]

```

12 //level order traversal ko hi tweek krna h isme
13
14
15 class Solution {
16 public:
17     vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
18         vector<vector<int>> ans;
19         if(root == nullptr) return ans;
20
21         queue<TreeNode*> q;
22         q.push(root);
23         int direction=1; //Left to right
24         //direction = 1 ; //right to left
25         while(!q.empty()){
26             int n=q.size();
27             vector<int> cur;
28             for(int i=0;i<n;i++){
29                 TreeNode* tem=q.front();
30                 q.pop();
31                 if(tem->left != nullptr){ q.push(tem->left),cur.push_back(tem->left->val); }
32                 if(tem->right != nullptr){ q.push(tem->right),cur.push_back(tem->right->val); }
33             }
34             if(direction==1){
35                 for(int i=0;i<cur.size()/2;i++){
36                     int swap=cur[i];
37                     cur[i]=cur[cur.size()-i-1];
38                     cur[cur.size()-i-1]=swap;
39                 }
40             }
41             if(cur.size()>0) ans.push_back(cur);
42             direction=1-direction;
43         }
44     }
45     return ans;
46 }
  
```

### L20. Boundary Traversal in Binary Tree

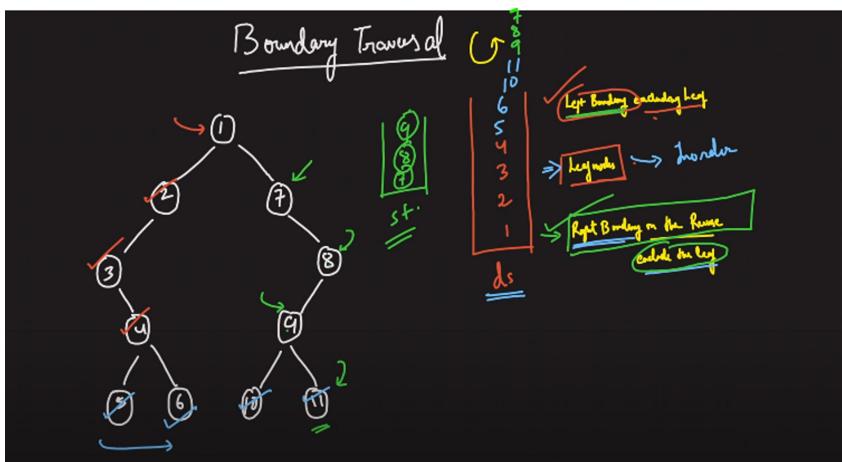
- <https://xiaoguan.gitbooks.io/leetcode/content/LeetCode/545-boundary-of-binary-tree-medium.html>
- <https://leetcode.com/problems/boundary-of-binary-tree/>
- <https://practice.geeksforgeeks.org/problems/boundary-traversal-of-binary-tree/1#>
- [https://www.codingninjas.com/codestudio/problems/boundary-traversal\\_790725](https://www.codingninjas.com/codestudio/problems/boundary-traversal_790725)

{https://app.gitbook.com/}

Left Boundary excluding leaf

Leaf nodes

Right Boundary on the Reverse  
include the leaf



L20. Boundary Traversal in Binary Tree | C++ | Java

```

C++   * Autocomplete
12 void addLeftBoundary(Node* root, vector<int> &res) {
13     Node* cur = root->left;
14     while (cur) {
15         if (!isLeaf(cur)) res.push_back(cur->data);
16         if (cur->left) cur = cur->left;
17         else cur = cur->right;
18     }
19 }
20 void addRightBoundary(Node* root, vector<int> &res) {
21     Node* cur = root->right;
22     vector<int> tmp;
23     while (cur) {
24         if (!isLeaf(cur)) tmp.push_back(cur->data);
25         if (cur->right) cur = cur->right;
26         else cur = cur->left;
27     }
28     for (int i = tmp.size()-1; i >= 0; --i) {
29         res.push_back(tmp[i]);
30     }
31 }
32
33 void addLeaves(Node* root, vector<int>& res) {
34     if (isLeaf(root)) {
35         res.push_back(root->data);
36         return;
37     }
38     if (root->left) addLeaves(root->left, res);
39     if (root->right) addLeaves(root->right, res);
40 }
41 public:
42     vector<int> printBoundary(Node *root)
43     {
44         vector<int> res;
45         if (!root) return res;
46         if (!isLeaf(root)) res.push_back(root->data);
47         addLeftBoundary(root, res);
48         addLeaves(root, res);
49         addRightBoundary(root, res);
50         return res;
51     }
52 }

```

```

Java   * Autocomplete
6
7 void addLeftBoundary(Node root, ArrayList<Integer> res) {
8     Node cur = root.left;
9     while (cur != null) {
10         if (!isLeaf(cur)) res.add(cur.data);
11         if (cur.left != null) cur = cur.left;
12         else cur = cur.right;
13     }
14 }
15 void addRightBoundary(Node root, ArrayList<Integer> res) {
16     Node cur = root.right;
17     ArrayList<Integer> tmp = new ArrayList<Integer>();
18     while (cur != null) {
19         if (!isLeaf(cur)) tmp.add(cur.data);
20         if (cur.right != null) cur = cur.right;
21         else cur = cur.left;
22     }
23     int i;
24     for (i = tmp.size()-1; i >= 0; --i) {
25         res.add(tmp.get(i));
26     }
27 }
28
29 void addLeaves(Node root, ArrayList<Integer> res) {
30     if (isLeaf(root)) {
31         res.add(root.data);
32         return;
33     }
34     if (root.left != null) addLeaves(root.left, res);
35     if (root.right != null) addLeaves(root.right, res);
36 }
37 ArrayList<Integer> printBoundary(Node node)
38 {
39     ArrayList<Integer> ans = new ArrayList<Integer>();
40     if (isLeaf(node)) ans.add(node.data);
41     addLeftBoundary(node, ans);
42     addLeaves(node, ans);
43     addRightBoundary(node, ans);
44     return ans;
45 }
46

```

T.C →  
 $O(H) + O(H) + O(N)$   
 $O(N) \leftarrow O(N)$   
 $SC \rightarrow O(N)$

$O(h)$  -> for left boundary of tree

$O(H)$  -> for right boundary of tree

$O(N)$  -> for pre-Order traversal for leaf nodes (the catch here is it has to be pre-Order, as any other order will not maintain the left to right nature of leaf nodes)

My Code : <https://practice.geeksforgeeks.org/problems/boundary-traversal-of-binary-tree/1#>

```

// { Driver Code Starts
#include <bits/stdc++.h>
using namespace std;
#define MAX_HEIGHT 100000

// Tree Node
struct Node
{
    int data;
    Node* left;
    Node* right;
};

// Utility function to create a new Tree Node
Node* newNode(int val)
{
    Node* temp = new Node;
    temp->data = val;
    temp->left = NULL;
    temp->right = NULL;

    return temp;
}

// Function to Build Tree
Node* buildTree(string str)
{
    // Corner Case

```

```

if(str.length() == 0 || str[0] == 'N')
    return NULL;

// Creating vector of strings from input
// string after splitting by space
vector<string> ip;

istringstream iss(str);
for(string str; iss >> str; )
    ip.push_back(str);

// Create the root of the tree
Node* root = newNode(stoi(ip[0]));

// Push the root to the queue
queue<Node*> queue;
queue.push(root);

// Starting from the second element
int i = 1;
while(!queue.empty() && i < ip.size()) {

    // Get and remove the front of the queue
    Node* currNode = queue.front();
    queue.pop();

    // Get the current node's value from the string
    string currVal = ip[i];

    // If the left child is not null
    if(currVal != "N") {

        // Create the left child for the current node
        currNode->left = newNode(stoi(currVal));

        // Push it to the queue
        queue.push(currNode->left);
    }

    // For the right child
    i++;
    if(i >= ip.size())
        break;
    currVal = ip[i];

    // If the right child is not null
    if(currVal != "N") {

        // Create the right child for the current node
        currNode->right = newNode(stoi(currVal));

        // Push it to the queue
        queue.push(currNode->right);
    }
    i++;
}

return root;
}
// } Driver Code Ends

/* A binary tree Node
struct Node
{
    int data;
    Node* left, * right;
}; */

class Solution {
public:
//17 N 1 N 9 N 2 N 11 N 16 N 8 N 5 N 3 N 10 N 13 N 5 N 13 N 7 N 10 N 4 N 7 N 2 N 12 N 16
//16 6 3 N 4 6 5 5 N 3 6 1 5 N 1 4 1 4 N 2 1 4 2
//4 10 N 5 5 N 6 7 N 8 8 N 8 11 N 3 4 N 1 3 N 8 6 N 11 11 N 5 8
    vector <int> boundary(Node *root)
{

```

```

if(root==nullptr) return {};

if(root->left == root->right && root->right == nullptr ){
    return {root->data};
}
//Your code here
vector<int> right,leaf,left,ans;
ans.push_back(root->data);
leftBoundary(root->left,left);
rightBoundary(root->right,right);
leafNodes(root,leaf);

for(int i=0;i<left.size();i++) ans.push_back(left[i]);//ans=left;
for(int i=0;i<leaf.size();i++) ans.push_back(leaf[i]);
for(int i=right.size()-1;i>=0;i--) ans.push_back(right[i]);

return ans;
}

bool isLeaf(Node *node){
    if(node->left == node->right && node->right == nullptr ) return true;
    return false;
}

void leafNodes(Node* node,vector<int> &leaf){
    if(node==nullptr) return;
    if(isLeaf(node)) leaf.push_back(node->data);//leaf node

    leafNodes(node->left,leaf);
    leafNodes(node->right,leaf);
}

void leftBoundary(Node* nodeLeft,vector<int> &left){
    if(nodeLeft==nullptr) return;
    if(isLeaf(nodeLeft)) return;//leaf node

    left.push_back(nodeLeft->data);
    if(nodeLeft->left!=nullptr) leftBoundary(nodeLeft->left,left);
    else leftBoundary(nodeLeft->right,left);

}

void rightBoundary(Node* nodeRight,vector<int> &right){
    if(nodeRight==nullptr) return;
    if(isLeaf(nodeRight)) return;//leaf node

    right.push_back(nodeRight->data);
    if(nodeRight->right!=nullptr) rightBoundary(nodeRight->right,right);
    else rightBoundary(nodeRight->left,right);

}

};

// { Driver Code Starts.

/* Driver program to test size function*/

int main() {
    int t;
    string tc;
    getline(cin, tc);
    t=stoi(tc);
    while(t--)
    {
        string s ,ch;
        getline(cin, s);
        Node* root = buildTree(s);
        Solution ob;
        vector <int> res = ob.boundary(root);
        for (int i : res) cout << i << " ";
        cout << endl;
    }
    return 0;
} // } Driver Code Ends

```

---

## L21. Vertical Order Traversal\*\*

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

987. Vertical Order Traversal of a Binary Tree

Hard 2260 2886 Add to List Share

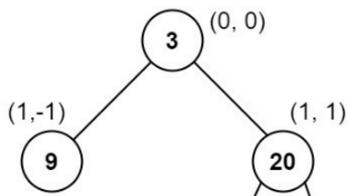
Given the root of a binary tree, calculate the **vertical order traversal** of the binary tree.

For each node at position  $(\text{row}, \text{col})$ , its left and right children will be at positions  $(\text{row} + 1, \text{col} - 1)$  and  $(\text{row} + 1, \text{col} + 1)$  respectively. The root of the tree is at  $(0, 0)$ .

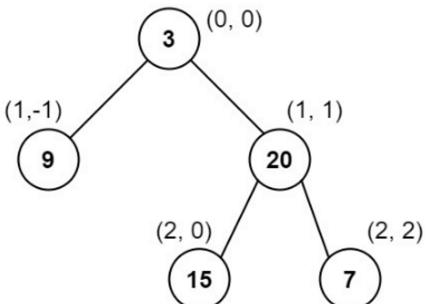
The **vertical order traversal** of a binary tree is a list of top-to-bottom orderings for each column index starting from the leftmost column and ending on the rightmost column. There may be multiple nodes in the same row and same column. In such a case, sort these nodes by their values.

Return the **vertical order traversal** of the binary tree.

**Example 1:**



**Example 1:**



Input: root = [3,9,20,null,null,15,7]

Output: [[9],[3,15],[20],[7]]

Explanation:

Column -1: Only node 9 is in this column.

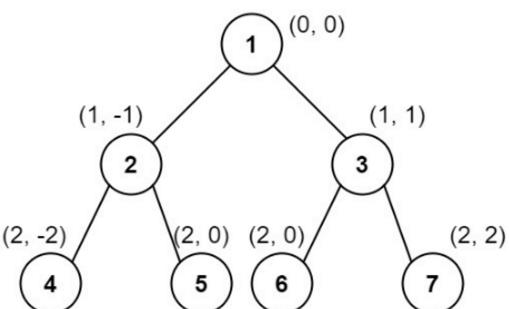
Column 0: Nodes 3 and 15 are in this column in that order from top to bottom.

Column 1: Only node 20 is in this column.

Column 2: Only node 7 is in this column.

**Example 2:**

**Example 2:**



Input: root = [1,2,3,4,5,6,7]

Output: [[4],[2],[1,5,6],[3],[7]]

Explanation:

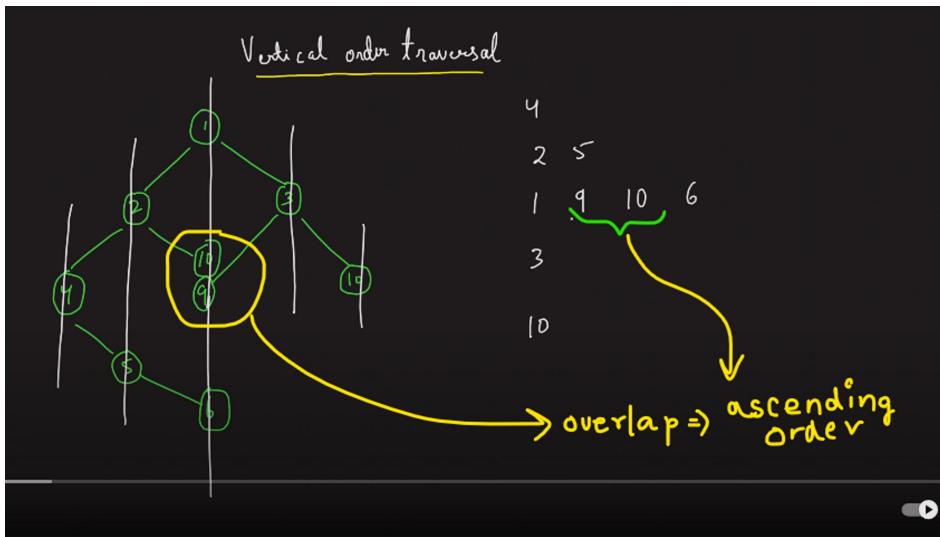
Column -2: Only node 4 is in this column.

Column -1: Only node 2 is in this column.

Column 0: Nodes 1, 5, and 6 are in this column.

1 is at the top, so it comes first.

5 and 6 are at the same position (2, 0), so we order them by their value, 5 before 6.



L21. Vertical Order Traversal of Binary Tree | C++ | Java

```
C++ class Solution {
public:
    vector<vector<int>> verticalTraversal(TreeNode* root) {
        map<int, map<int, multiset<int>>> nodes;
        queue<pair<TreeNode*, pair<int, int>>> todo;
        todo.push({root, {0, 0}});
        while (!todo.empty()) {
            auto p = todo.front();
            todo.pop();
            TreeNode* node = p.first;
            int x = p.second.first, y = p.second.second;
            nodes[x][y].insert(node->val);
            if (node->left) {
                todo.push({node->left, {x - 1, y + 1}});
            }
            if (node->right) {
                todo.push({node->right, {x + 1, y + 1}});
            }
        }
        vector<vector<int>> ans;
        for (auto p : nodes) {
            vector<int> col;
            for (auto q : p.second) {
                col.insert(col.end(), q.second.begin(), q.second.end());
            }
            ans.push_back(col);
        }
        return ans;
    }
};

Java
class Tuple {
    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<Tuple>();
        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;

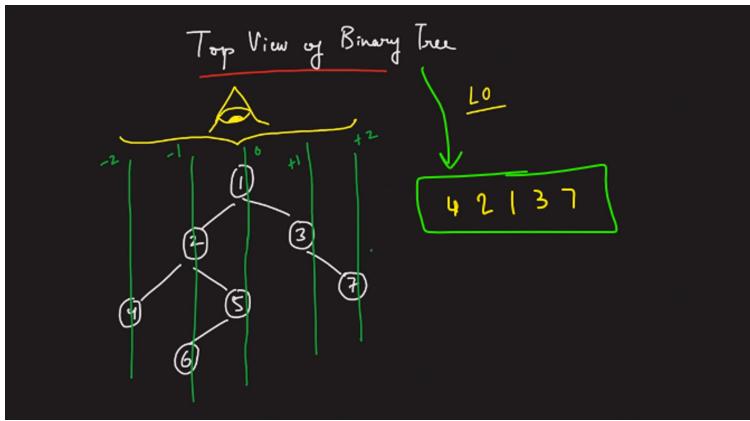
            if (!map.containsKey(x)) {
                map.put(x, new TreeMap<>());
            }
            if (!map.get(x).containsKey(y)) {
                map.get(x).put(y, new PriorityQueue<>());
            }
            map.get(x).get(y).offer(node.val);

            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()) {
                    System.out.println(nodes.peek());
                    list.get(list.size() - 1).add(nodes.poll());
                }
            }
        }
    }
}
```

- Map<int, map<int, multiset<int>>> nodes => this is :: <x,<y,{set of integers that overlap in sorted order }>>
- Also note that we use multiset as:
  - 2 or more nodes can have same value
  - we want overlapping nodes in ascending order
- Queue<pair<TreeNode\*, pair<int, int>>> => this stores :: <TreeNode\* node ,{x,y}>
- X,y are co-ordinates of of that node
- Queue is used for level-order traversal
- Vector col :: stores all the nodes on the same vertical line!

## L22. Top view of binary tree

<https://www.geeksforgeeks.org/print-nodes-top-view-binary-tree/>  
<https://www.interviewbit.com/blog/top-view-of-binary-tree/>



```
C++ Autocomplete
L22. Top View of Binary Tree | C++ | Java

1 class Solution
2 {
3     public:
4         //Function to return a list of nodes visible from the top view
5         //from left to right in Binary Tree.
6         vector<int> topView(Node *root)
7     {
8         vector<int> ans;
9         if(root == NULL) return ans;
10        map<int,int> mpp;
11        queue<pair<Node*, int>> q;
12        q.push({root, 0});
13        while(!q.empty()) {
14            auto it = q.front();
15            q.pop();
16            Node* node = it.first;
17            int line = it.second;
18            if(mpp.find(line) == mpp.end()) mpp[line] = node->data;
19
20            if(node->left != NULL) {
21                q.push({node->left, line-1});
22            }
23            if(node->right != NULL) {
24                q.push({node->right, line + 1});
25            }
26        }
27
28        for(auto it : mpp) {
29            ans.push_back(it.second);
30        }
31        return ans;
32    }
33
34
35    for(auto it : mpp) {
36        ans.push_back(it.second);
37    }
38    return ans;
39}
40
41};

Java Autocomplete
i l <
6 _hd;
7 _node;
8 }
9 }
10
11 class Solution
12 {
13     //Function to return a list of nodes visible from the top view
14     //from left to right in Binary Tree.
15
16
17     public static ArrayList<Integer> topView(Node root)
18     {
19         ArrayList<Integer> ans = new ArrayList<()>;
20         if(root == null) return ans;
21         Map<Integer, Integer> map = new TreeMap<()>;
22         Queue<Pair> q = new LinkedList<Pair>();
23         q.add(new Pair(root, 0));
24         while(!q.isEmpty()) {
25             Pair it = q.remove();
26             int hd = it.hd;
27             Node temp = it.node;
28             if(map.get(hd) == null) map.put(hd, temp.data);
29             if(temp.left != null) {
30
31                 q.add(new Pair(temp.left, hd - 1));
32             }
33             if(temp.right != null) {
34
35                 q.add(new Pair(temp.right, hd + 1));
36             }
37         }
38
39         for (Map.Entry<Integer, Integer> entry : map.entrySet()) {
40             ans.add(entry.getValue());
41         }
42         return ans;
43     }
44
45 }
46 }

TUF
```

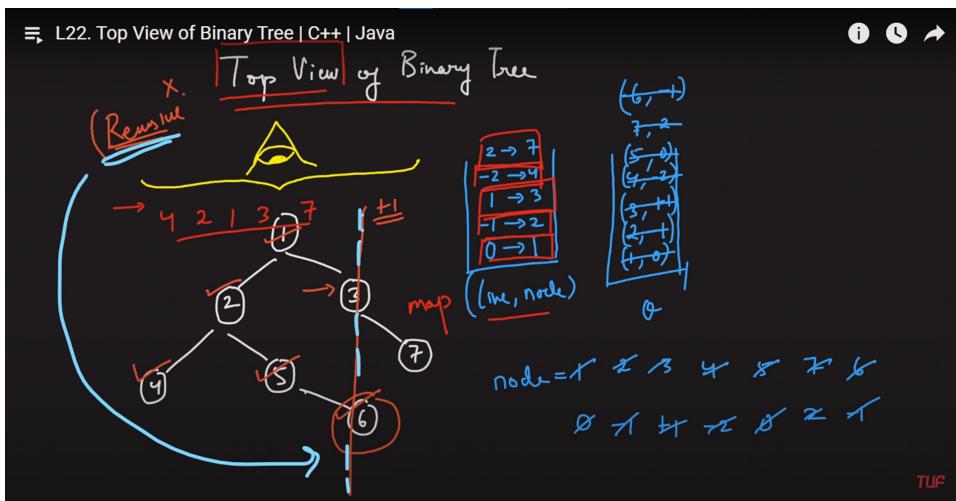
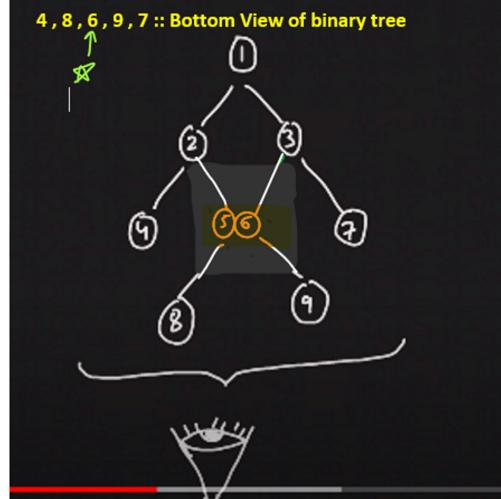
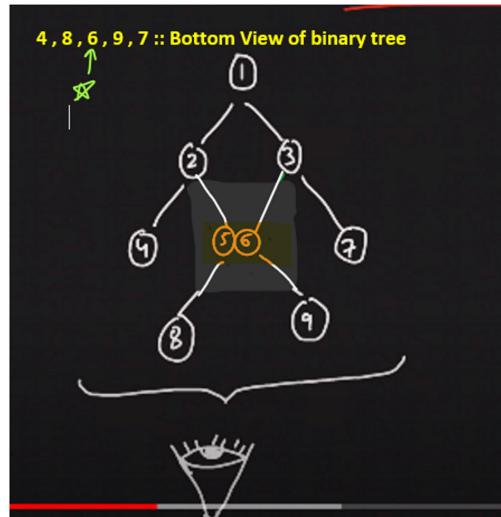
**Question :** Can we do the top view of binary tree using some recursive traversal ?

**Answer :** Yes, we can BUT, *there's a catch*, we cannot simply use the above algo, but we have to add/consider another parameter of:

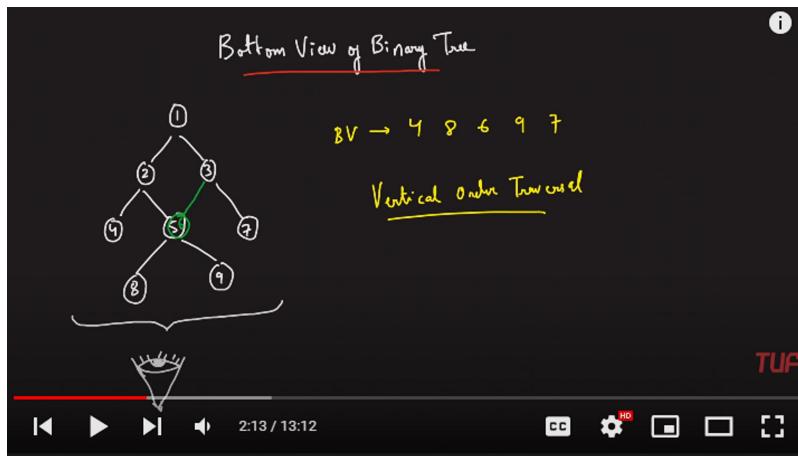
- height (depth of current node from the root node)

Since in recursive traversal it may happen that the lower node is visited before the upper node and then that ruins the ans , so we have to maintain on the same x-co-ordinate , which is the minimum height node and add that node to the top view ans for that x-coordinate

So for this question prefer level-order traversal as its easier to implement as here we don't have to keep track of the depth of the node. But if we implement using recursive traversal then we have to implement the extra logic of depth of node.

L23. Bottom View of binary tree

**Note:** For overlapping nodes at same co-ordinate we take the one to the right for that coordinate  
EG: node 5 & 6 overlap , so we take 6 for that position!



```

C++ L23. Bottom View of Binary Tree | C++ | Java
class Solution {
public:
    vector<int> bottomView(Node *root) {
        vector<int> ans;
        if(root == NULL) return ans;
        map<int,int> mpp;
        queue<pair<Node*, int>> q;
        q.push({root, 0});
        while(!q.empty()) {
            auto it = q.front();
            q.pop();
            Node* node = it.first;
            int line = it.second;
            mpp[line] = node->data;
            if(node->left != NULL) {
                q.push({node->left, line-1});
            }
            if(node->right != NULL) {
                q.push({node->right, line + 1});
            }
        }
        for(auto it : mpp) {
            ans.push_back(it.second);
        }
        return ans;
    }
};

Java
class Solution {
    //Function to return a list containing the bottom view of the given tree.
    public ArrayList <Integer> bottomView(Node root)
    {
        ArrayList<Integer> ans = new ArrayList<>();
        if(root == null) return ans;
        Map<Integer, Integer> map = new TreeMap<>();
        Queue<Node> q = new LinkedList<Node>();
        root.hd = 0;
        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> entry : map.entrySet()) {
            ans.add(entry.getValue());
        }
        return ans;
    }
}

```

My Code : with pbm link : <https://practice.geeksforgeeks.org/problems/bottom-view-of-binary-tree/1#>

### Bottom View of Binary Tree

**Medium** Accuracy: 45.32% Submissions: 91790 Points: 4

Given a binary tree, print the bottom view from left to right.  
A node is included in bottom view if it can be seen when we look at the tree from bottom.

```

20
 / \
8   22
/ \   \
5   3   25
 / \
10  14

```

For the above tree, the bottom view is 5 10 3 14 25.

If there are multiple bottom-most nodes for a horizontal distance from root, then print the later one in level traversal. For example, in the below diagram, 3 and 4 are both the bottommost nodes at horizontal distance 0, we need to print 4.

20

```

//Function to return a list containing the bottom view of the given tree.
94
95
96 class Solution {
97     public:
98     vector <int> bottomView(Node *root) {
99         // Your Code Here
100        vector<int> ans;
101        if(root==nullptr) return ans;
102        queue<pair<Node*, int>> q; // node,x
103        map<int,int> node;//x,data
104        int n=1e5+10,x;
105        vector<int> vis(n,0);
106        q.push({root,0});
107        while(!q.empty()){
108            Node* tem=q.front().first; x=q.front().second;
109            node[x]=tem->data;
110            q.pop();
111            if(tem->left!=nullptr) q.push({tem->left,x-1});
112            if(tem->right!=nullptr) q.push({tem->right,x+1});
113        }
114        for(auto x:node){
115            ans.push_back(x.second);
116        }
117    }
118 };
119

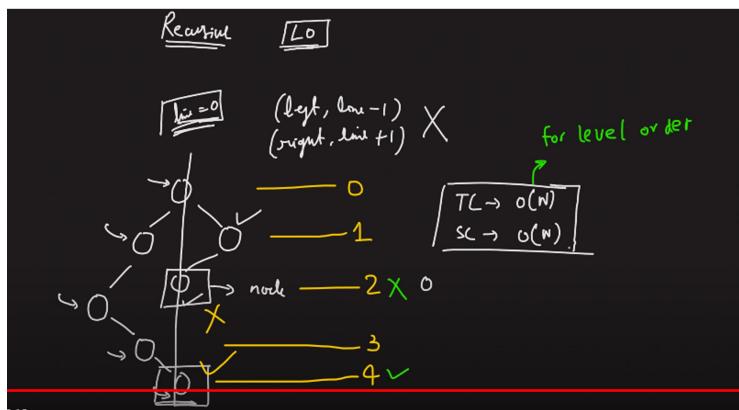
```

Question: Will recursive traversal work for this question?

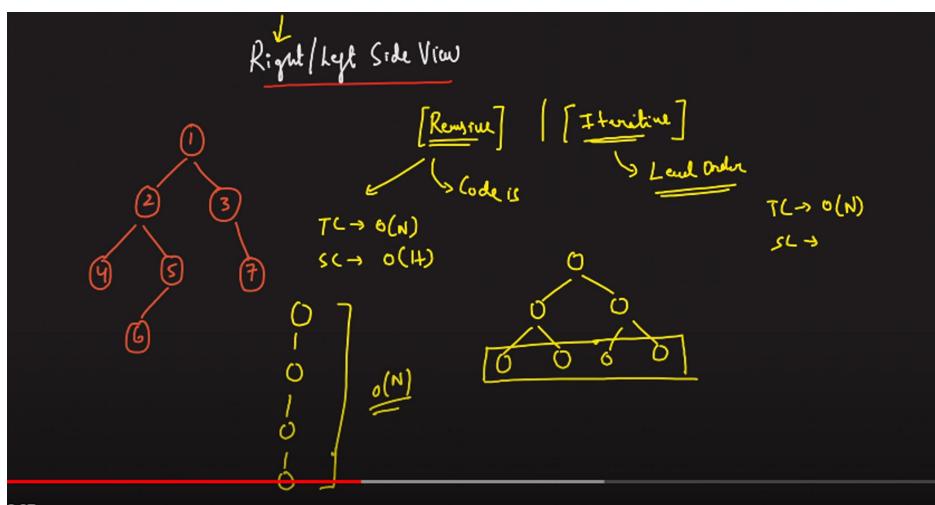
Answer : Yes it will work BUT not directly.

We will have to consider the depth of the current node wrt the root and the highest depth node will be the answer for a x-co-ordinate! So for this question prefer level-order traversal as its easier to implement as here we don't have to keep track of the depth of the node

But if we implement using recursive traversal then we have to implement the extra logic of depth of node.



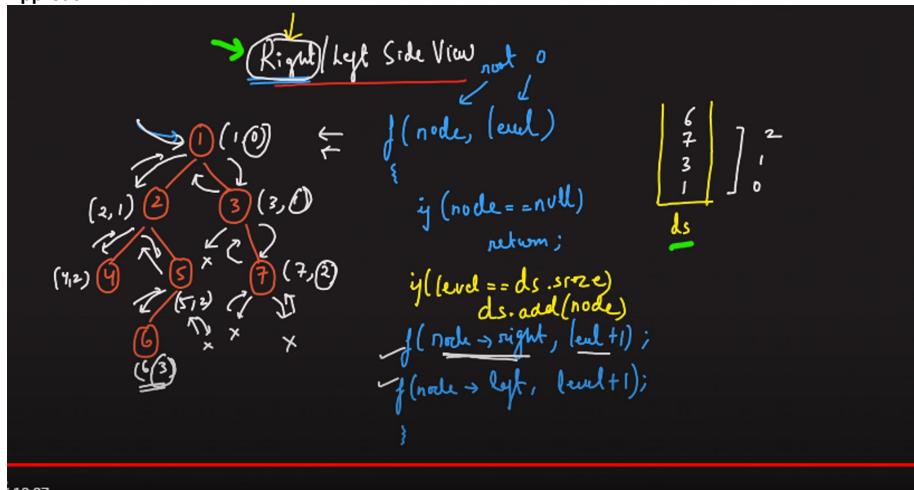
#### L24. Right Side View of Binary Tree



#### 2 Approaches:

- 1) Recursive Traversal (smarter solution): We use reverse Pre-order Traversal : i.e, Root,Right,Left (for right side view)
  - TC :  $O(N)$  , Auxillary SC =  $O(\text{Height of tree})$
  - Smart thing : here we check if current ans.size() equals the level we are on, which ensures we put only 1 element on each level
  - To ensure it's the right most element of that level we visit first, we traverse in the following order: root -> right -> left
- 2) Iterative Traversal : Level Order Traversal
  - TC :  $O(N)$  , Auxillary SC =  $O(N)$

#### Approach 1:



L24. Right/Left View of Binary Tree | C++ | Java

```

1  /**
2   * Definition for a binary tree node.
3   */
4   struct TreeNode {
5     int val;
6     TreeNode *left;
7     TreeNode *right;
8   };
9   TreeNode() : val(0), left(nullptr), right(nullptr) {}
10  TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
11  TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
12}
13
14
15
16
17 class Solution {
18 public:
19 void rightSideView(TreeNode *root) {
20   vector<int> res;
21   recursion(root, 0, res);
22   return res;
23 }
24 void recursion(TreeNode *root, int level, vector<int> &res)
25 {
26   if(root==NULL) return ;
27   if(res.size()==level) res.push_back(root->val);
28   recursion(root->right, level+1, res);
29   recursion(root->left, level+1, res);
30 }
31
32 };
33
34
35

```

Java

```

1 /**
2  * Definition for a binary tree node.
3  */
4  public class TreeNode {
5   int val;
6   TreeNode left;
7   TreeNode right;
8  };
9  TreeNode(int val) { this.val = val; }
10 TreeNode(int val, TreeNode left, TreeNode right) {
11   this.val = val;
12   this.left = left;
13   this.right = right;
14 }
15
16 public class Solution {
17 public List<Integer> rightSideView(TreeNode root) {
18   List<Integer> result = new ArrayList<Integer>();
19   rightView(root, result, 0);
20   return result;
21 }
22
23 public void rightView(TreeNode curr, List<Integer> result, int currDepth){
24   if(curr == null){
25     return;
26   }
27   if(currDepth == result.size()){
28     result.add(curr.val);
29   }
30   rightView(curr.right, result, currDepth + 1);
31   rightView(curr.left, result, currDepth + 1);
32 }
33
34 }
35

```

TUF

## Approach 2:

### My Approach : Level Order Traversal

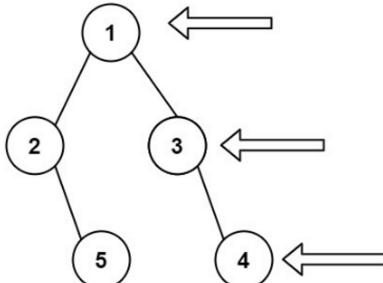
TC : O(N) , Auxillary SC = O(N)

199. Binary Tree Right Side View

Medium 5261 287 Add to List Share

Given the `root` of a binary tree, imagine yourself standing on the **right side** of it, return the *values of the nodes you can see ordered from top to bottom*.

**Example 1:**



Input: root = [1,2,3,null,5,null,4]  
Output: [1,3,4]

**Example 2:**

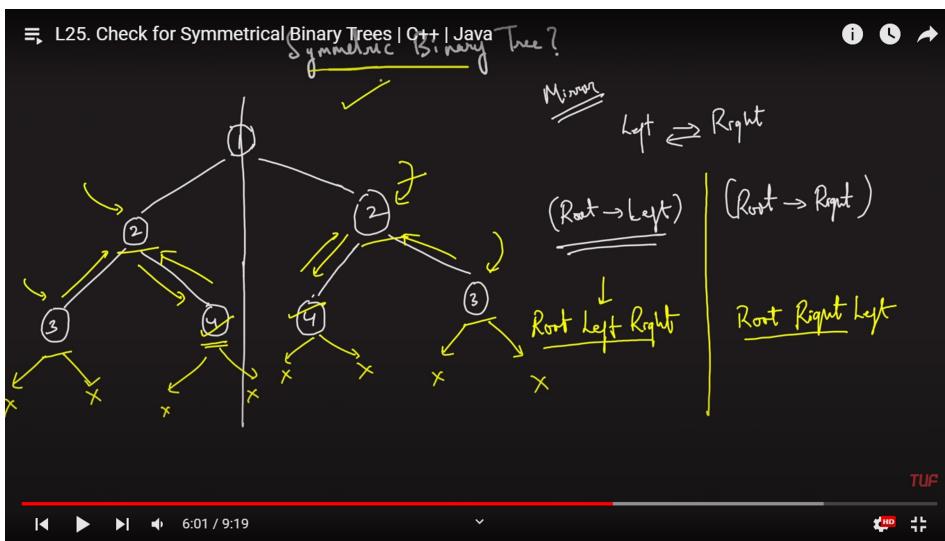
## L25. Symmetric Tree

TC= O(N) ,SC=O(N) (assuming skewed tree)

```

1 /**
2  * Definition for a binary tree node.
3  */
4  struct TreeNode {
5   int val;
6   TreeNode *left;
7   TreeNode *right;
8  };
9  TreeNode() : val(0), left(nullptr), right(nullptr) {}
10 TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
11 TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
12}
13
14
15
16
17 class Solution {
18 public:
19 vector<int> rightSideView(TreeNode* root) {
20   //at each level , the x-max
21   //level-order traversal =>last element to be entered
22   vector<int> ans;
23   if(root==nullptr) return ans;
24   queue<TreeNode*> q;
25   q.push(root);
26   while(!q.empty()){
27     int n = q.size();
28     for(int i=0;i<n;i++){
29       if(i==n-1) ans.push_back(q.front()->val); //for right side view
30       // if(i==0) ans.push_back(q.front()->val); //for left side view
31       TreeNode* tem=q.front();
32       q.pop();
33       if(tem->left != nullptr) q.push(tem->left);
34       if(tem->right != nullptr) q.push(tem->right);
35     }
36   }
37   return ans;
38 }
39

```



L25. Check for Symmetrical Binary Trees | C++ | Java

```

1+ /**
2+ * Definition for a binary tree node.
3+ * struct TreeNode {
4+ *     int val;
5+ *     TreeNode *left;
6+ *     TreeNode *right;
7+ *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
8+ *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
9+ *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
10+ * };
11+ */
12+ class Solution {
13+ public:
14+     bool isSymmetric(TreeNode* root) {
15+         return root==NULL || isSymmetricHelp(root->left, root->right);
16+     }
17+     bool isSymmetricHelp(TreeNode* left, TreeNode* right){
18+         if(left==NULL || right==NULL)
19+             return left==right;
20+
21+         if(left.val!=right.val) return false;
22+
23+         return isSymmetricHelp(left.left, right.right)
24+             &&
25+             isSymmetricHelp(left.right, right.left);
26+     }
27+ };

```

```

1+ /**
2+ * Definition for a binary tree node.
3+ * public class TreeNode {
4+ *     int val;
5+ *     TreeNode left;
6+ *     TreeNode right;
7+ *     TreeNode() {}
8+ *     TreeNode(int val) { this.val = val; }
9+ *     TreeNode(int val, TreeNode left, TreeNode right) {
10+ *         this.val = val;
11+ *         this.left = left;
12+ *         this.right = right;
13+ *     }
14+ * }
15+ */
16+ class Solution {
17+ public boolean isSymmetric(TreeNode root) {
18+     return root==null || isSymmetricHelp(root.left, root.right);
19+ }
20+
21+ private boolean isSymmetricHelp(TreeNode left, TreeNode right){
22+     if(left==null || right==null)
23+         return left==right;
24+
25+     if(left.val!=right.val) return false;
26+
27+     return isSymmetricHelp(left.left, right.right)
28+         &&
29+         isSymmetricHelp(left.right, right.left);
30+ }
31+
32+

```

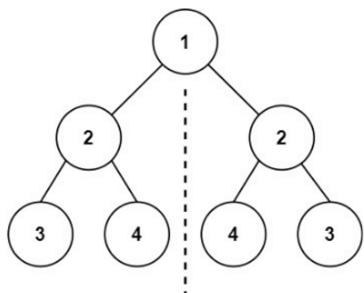
My Approach(same as above , just slightly different implementation) :

### 101. Symmetric Tree

Easy ⚡ 7823 ⌂ 193 Add to List Share

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

```

1+ /**
2+ * Definition for a binary tree node.
3+ * struct TreeNode {
4+ *     int val;
5+ *     TreeNode *left;
6+ *     TreeNode *right;
7+ *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
8+ *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
9+ *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
10+ * };
11+ */
12+ class Solution {
13+ public:
14+ // TC=O(n) , SC=O(n)
15+     bool isSymmetric(TreeNode* root) {
16+         bool ans=true;
17+         mirror(root->left,root->right,ans);
18+         return ans;
19+     }
20+     void mirror(TreeNode* left,TreeNode* right,bool& ans){
21+         if(left==nullptr && right==nullptr) return;
22+         if(left==nullptr || right==nullptr){ ans=false;return; }
23+         if(left->val!=right->val){ ans=false;return; }
24+
25+         mirror(left->left,right->right,ans);
26+         mirror(left->right,right->left,ans);
27+     }
28+

```

Your previous code was restored from your local storage. [Reset to default](#)

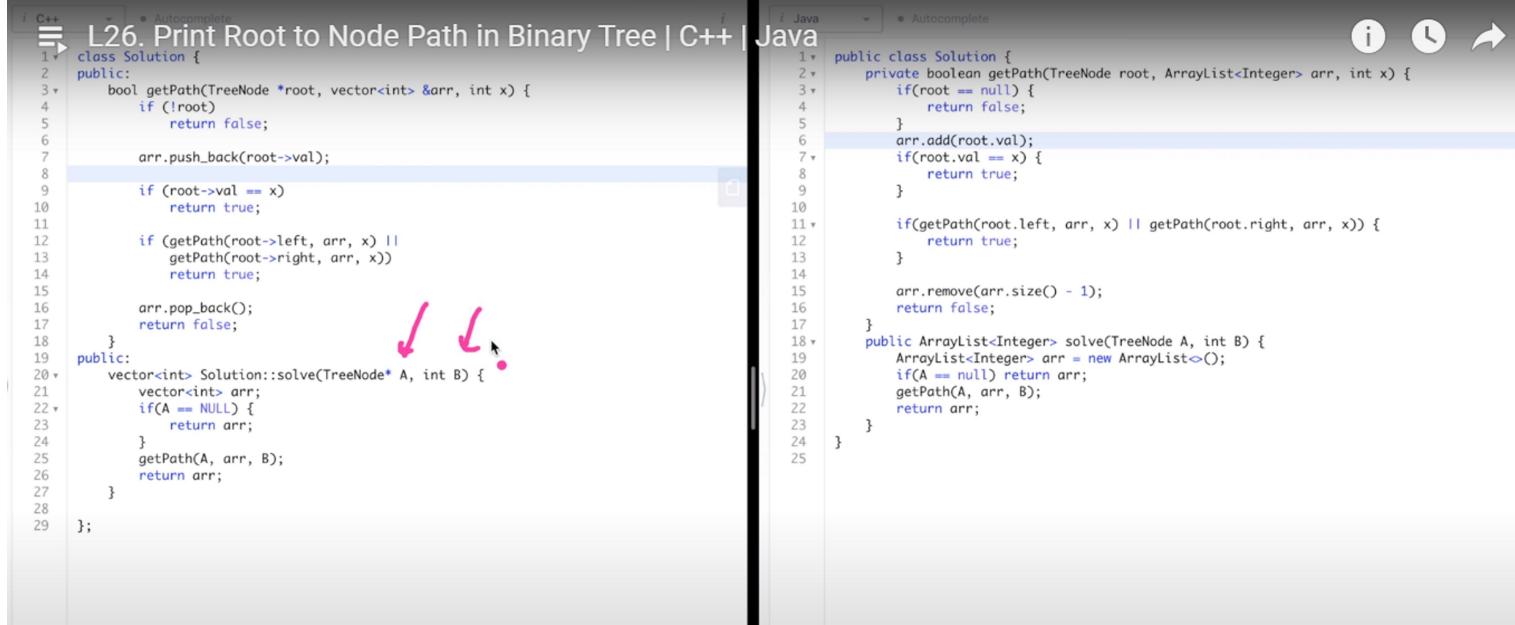
---

## L26. Print Path from Root to Node (given the path exists)

---

Question Link: <https://www.interviewbit.com/problems/path-to-given-node/>

Answer:



```
C++ // Autocomplete
1 class Solution {
2 public:
3     bool getPath(TreeNode *root, vector<int> &arr, int x) {
4         if (!root)
5             return false;
6
7         arr.push_back(root->val);
8
9         if (root->val == x)
10            return true;
11
12         if (getPath(root->left, arr, x) || 
13             getPath(root->right, arr, x))
14            return true;
15
16         arr.pop_back();
17     }
18
19     public:
20     vector<int> Solution::solve(TreeNode* A, int B) {
21         vector<int> arr;
22         if(A == NULL) {
23             return arr;
24         }
25         getPath(A, arr, B);
26         return arr;
27     }
28 };
29 
```

```
Java // Autocomplete
1 public class Solution {
2     private boolean getPath(TreeNode root, ArrayList<Integer> arr, int x) {
3         if(root == null) {
4             return false;
5         }
6         arr.add(root.val);
7         if(root.val == x) {
8             return true;
9         }
10
11         if(getPath(root.left, arr, x) || getPath(root.right, arr, x)) {
12             return true;
13         }
14
15         arr.remove(arr.size() - 1);
16     }
17
18     public ArrayList<Integer> solve(TreeNode A, int B) {
19         ArrayList<Integer> arr = new ArrayList<>();
20         if(A == null) return arr;
21         getPath(A, arr, B);
22         return arr;
23     }
24 }
25 
```

```
bool getPath(TreeNode *root, vector<int> &arr, int x) {
// if root is NULL
// there is no path
if (!root)
    return false;

// push the node's value in 'arr'
arr.push_back(root->val);

// if it is the required node
// return true
if (root->val == x)
    return true;

// else check whether the required node lies
// in the left subtree or right subtree of
// the current node
if (getPath(root->left, arr, x) || 
    getPath(root->right, arr, x))
    return true;

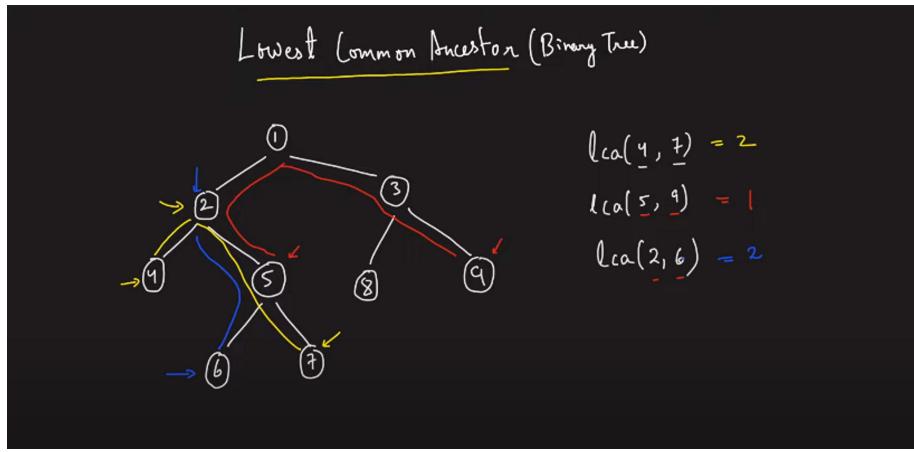
// required node does not lie either in the
// left or right subtree of the current node
// Thus, remove current node's value from
// 'arr' and then return false
arr.pop_back();
return false;
}

vector<int> Solution::solve(TreeNode* A, int B) {
    vector<int> arr;
    if(A == NULL) {
        return arr;
    }
    getPath(A, arr, B);
    return arr;
}
```

---

## L27.

---



L27. Lowest Common Ancestor in Binary Tree | LCA | C++ | Java

C++

```

1+ /**
2+ * Definition for a binary tree node.
3+ * struct TreeNode {
4+ *     int val;
5+ *     TreeNode *left;
6+ *     TreeNode *right;
7+ *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
8+ * };
9+
10+ class Solution {
11+ public:
12+     TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
13+         //base case
14+         if (root == NULL || root == p || root == q) {
15+             return root;
16+         }
17+         TreeNode* left = lowestCommonAncestor(root->left, p, q);
18+         TreeNode* right = lowestCommonAncestor(root->right, p, q);

19+         //result
20+         if(left == NULL) {
21+             return right;
22+         }
23+         else if(right == NULL) {
24+             return left;
25+         }
26+         else { //both left and right are not null, we found our result
27+             return root;
28+         }
29+     }
30+ };

```

Java

```

1+ /**
2+ * Definition for a binary tree node.
3+ * public class TreeNode {
4+ *     int val;
5+ *     TreeNode left;
6+ *     TreeNode right;
7+ *     TreeNode(int x) { val = x; }
8+ * };
9+
10+ class Solution {
11+ public:
12+     public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
13+         //base case
14+         if (root == null || root == p || root == q) {
15+             return root;
16+         }
17+         TreeNode left = lowestCommonAncestor(root.left, p, q);
18+         TreeNode right = lowestCommonAncestor(root.right, p, q);

19+         //result
20+         if(left == null) {
21+             return right;
22+         }
23+         else if(right == null) {
24+             return left;
25+         }
26+         else { //both left and right are not null, we found our result
27+             return root;
28+         }
29+     }
30+ };

```

TC=O(N) , ASC=O(N)

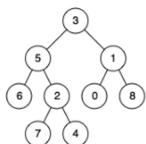
### 236. Lowest Common Ancestor of a Binary Tree

Medium 7783 233 Add to List Share

Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree.

According to the definition of LCA on Wikipedia: "The lowest common ancestor is defined between two nodes  $p$  and  $q$  as the lowest node in  $T$  that has both  $p$  and  $q$  as descendants (where we allow a node to be a descendant of itself)."

#### Example 1:



Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1

Output: 3

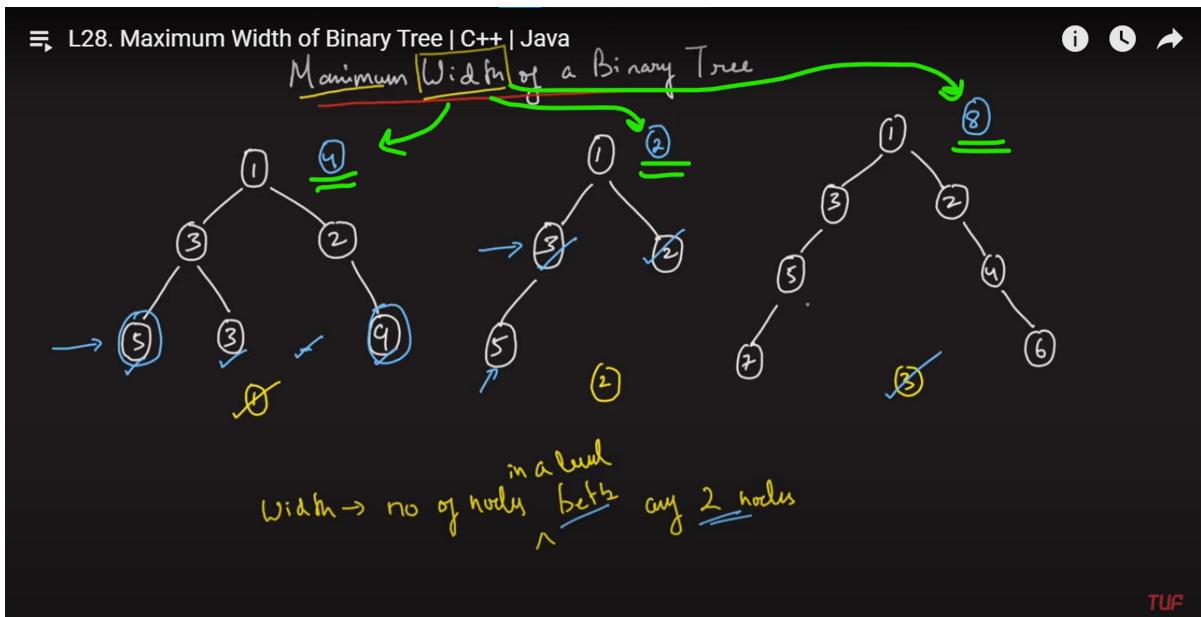
Explanation: The LCA of nodes 5 and 1 is 3.

#### Note :

- we can prune the backtrack if on a node we get both left and right as NOT NULL, and prune it when this type of node first occurs
- There can be another case where we HAVE to go to the root if only one node either left/right returns NOT NULL value
  - because then that is the case when one of the nodes (of the 2 nodes whose LCA is to be found) is the ancestor of ANOTHER
  - And in that case there will never be a node in tree for which both left and right will be NOT NULL, and hence the backtrack will end at root node
  - We will need to backtrack till root in this case to ensure that current node being returned is the DFA

=====

L28.  
Question link: <https://leetcode.com/problems/maximum-width-of-binary-tree/>

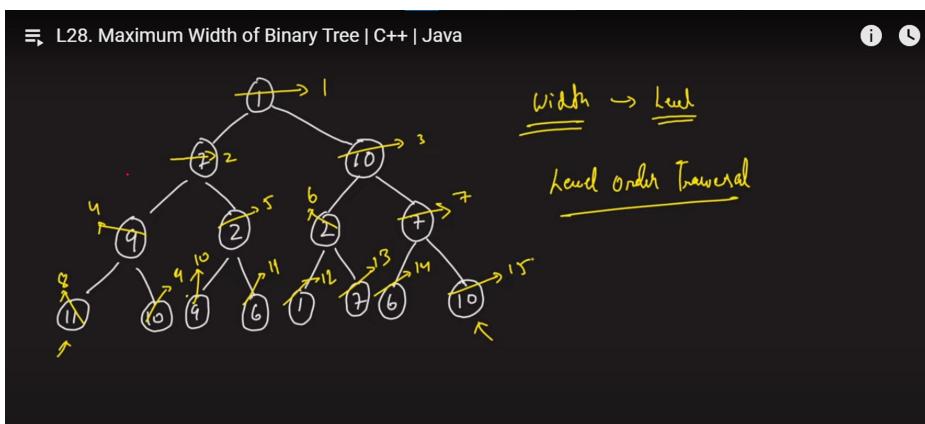


**Width of binary tree:** number of nodes in a level between the 2 extreme nodes present(including those nodes/extreme nodes).

- Note the extreme nodes must be present in the binary tree to count the nodes in between them(between nodes may/may not exist)

**Approach :**

- If we index the tree like shown in yellow below then the question becomes quite easy, at each level simply find the first and last index present which gives us the width at that level
- update max width at each level while doing level order traversal



**Striver's Code:**

L28. Maximum Width of Binary Tree | C++ | Java

```

C++ # Autocomplete
1 * Definition for a binary tree node.
2   struct TreeNode {
3     int val;
4     TreeNode *left;
5     TreeNode *right;
6   }
7   TreeNode() : val(0), left(nullptr), right(nullptr) {}
8   TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
9   TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
10  */
11
12 v class Solution {
13 public:
14   int widthOfBinaryTree(TreeNode* root) {
15     if(!root)
16       return 0;
17     int ans = 0;
18     queue<pair<TreeNode*, int>> q;
19     q.push({root, 0});
20     while(!q.empty()){
21       int size = q.size();
22       int mmin = q.front().second; //to make the id starting from zero
23       int first, last;
24       for(int i=0; i<size; i++){
25         int cur_id = q.front().second-mmin;
26         TreeNode* node = q.front().first;
27         q.pop();
28         if(i==0) first = cur_id;
29         if(i==size-1) last = cur_id;
30         if(node->left)
31           q.push({node->left, cur_id*2+1});
32         if(node->right)
33           q.push({node->right, cur_id*2+2});
34       }
35       ans = max(ans, last-first+1);
36     }
37   }
38   return ans;
39 }

```

```

Java # Autocomplete
9   * TreeNode(int val, TreeNode left, TreeNode right) {
10    *   this.val = val;
11    *   this.left = left;
12    *   this.right = right;
13    */
14
15 v class Pair {
16   TreeNode node;
17   int num;
18   Pair(TreeNode _node, int _num) {
19     num = _num;
20     node = _node;
21   }
22 }
23
24 v class Solution {
25 public:
26   int widthOfBinaryTree(TreeNode root) {
27     if(root == null) return 0;
28     int ans = 0;
29     Queue<Pair> q = new LinkedList<Pair>();
30     q.offer(new Pair(root, 0));
31     while(!q.isEmpty()){
32       int size = q.size();
33       int mmin = q.peek().num; //to make the id starting from zero
34       int first = 0, last = 0;
35       for(int i=0; i<size; i++){
36         int cur_id = q.peek().num-mmin;
37         TreeNode node = q.peek().node;
38         q.poll();
39         if(i==0) first = cur_id;
40         if(i==size-1) last = cur_id;
41         if(node.left != null)
42           q.offer(new Pair(node.left, cur_id*2+1));
43         if(node.right != null)
44           q.offer(new Pair(node.right, cur_id*2+2));
45       }
46       ans = Math.max(ans, last-first+1);
47     }
48   }
49 }

```

### My initial Solution:

#### My Code is easier to understand for me at least!

Note: the approach is correct here , but there will be an overflow error for indexing if the tree is skewed (max:  $2^{3000}$ , can be the index)

- so we have to find a way to index the nodes to avoid overflow
- We do that by
  - o Index : {index - leftmost\_index\_on\_that\_level}

### 662. Maximum Width of Binary Tree

Medium 2986 482 Add to List Share

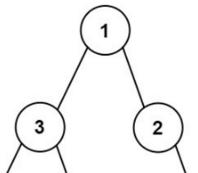
Given the root of a binary tree, return the **maximum width** of the given tree.

The **maximum width** of a tree is the maximum **width** among all levels.

The **width** of one level is defined as the length between the end-nodes (the leftmost and rightmost non-null nodes), where the null nodes between the end-nodes are also counted into the length calculation.

It is **guaranteed** that the answer will in the range of **32-bit** signed integer.

#### Example 1:



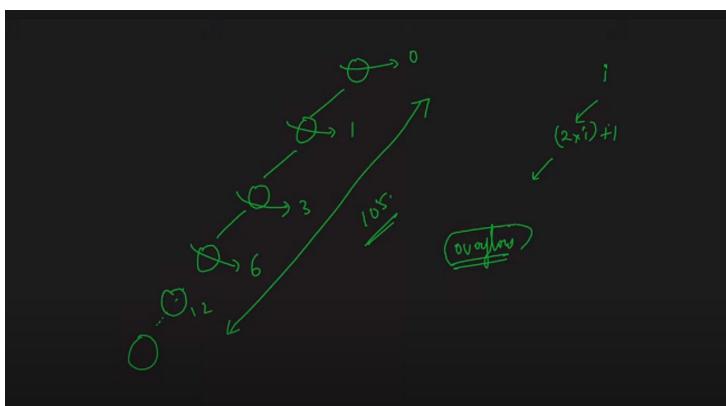
```

19 v // LEVEL ORDER (REVERSE), INDEX = 1,2,3,4,5,6,7,8,9,10,11,12,13,14
20 int widthOfBinaryTree(TreeNode* root) {
21   if(root==nullptr) return 0;
22   int width=1;
23   long long int first,last;
24   queue<pair<TreeNode*,long long>> q;
25   q.push({root,1});
26   while(!q.empty()){
27     int nq_size();
28     for(int i=0;i<nq_size();i++){
29       TreeNode* tem=q.front().first;
30       long long index=q.front().second;
31       q.pop();
32       if(i==0) ifirst=index;
33       if(i==nq_size()-1) ilast=index;
34
35       if(tem->left!=nullptr) q.push({tem->left,2*index});
36       if(tem->right!=nullptr) q.push({tem->right,2*index+1});
37     }
38     width=max(width,(int)(ilast-ifirst+1));
39   }
40 }

```

Runtime Error

Line 33: Char 61: runtime error: signed integer overflow: 2 \* 6917529027641081855 cannot be represented in type 'long long' (solution.cpp)  
SUMMARY: UndefinedBehaviorSanitizer: undefined-behavior prog\_main.cpp:42:61



One simple change in the code and the overflowing error is resolved :

leetcode.com/problems/maximum-width-of-binary-tree/submissions/

Apps Building Modern Web Guide to write an ar... Workspace Log in | Innovation... Problem setting gui... Apply | CodeChef cs50.harvard.edu Other bookmarks Reading list

LeetCode Explore Problems Interview Contest Discuss Store LeetCode Challenge + GIVEAWAY! Premium

Description Solution Discuss (918) Submissions C++ Autocomplete

**Success** Details >

Runtime: 7 ms, faster than 76.13% of C++ online submissions for Maximum Width of Binary Tree.

Memory Usage: 17.3 MB, less than 43.41% of C++ online submissions for Maximum Width of Binary Tree.

Next challenges:

- Max Area of Island
- Minimum Moves to Reach Target with Rotations
- Smallest Missing Genetic Value in Each Subtree

Show off your acceptance: [Facebook](#) [Twitter](#) [LinkedIn](#)

Time Submitted	Status	Runtime	Memory	Language
11/23/2021 12:17	Accepted	7 ms	17.3 MB	cpp
11/23/2021 11:53	Runtime Error	N/A	N/A	cpp

```

9 * TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
10 *
11 */
12 */
13 class Solution {
14 public:
15     // we index each node in level order traversal :1,2,3,4... using below formula
16     // {TreeNode* tem,index} : children :: {tem->left,2^index},{tem->right,2^index+1}
17     // Then find the width at each level , given by: last_i - first_i + 1
18     // level order traversal: last_i - first_i + 1
19     int widthOfBinaryTree(TreeNode* root) {
20         if(root==NULL) return 0;
21         int width=1;
22         long long ifirst,ilast;
23         queue<pair<TreeNode*,long long>> q;
24         q.push({root,1});
25         while(!q.empty()){
26             int n=q.size();
27             for(int i=0;i<n;i++){
28                 TreeNode* tem=q.front().first;
29                 long long index=q.front().second;
30                 q.pop();
31                 if(i==0) ifirst=index;
32                 if(i==n-1) ilast=index;
33                 index += ifirst; // ***TO AVOID OVERFLOWING INDEX***
34                 if(tem->left!=NULL) q.push({tem->left,2*index});
35                 if(tem->right!=NULL) q.push({tem->right,2*index+1});
36             }
37             width=max(width,(int)(ilast-ifirst+1));
38         }
39     }
40     return width;
41 }
42 }
```

## L29.

Video Link : [L29. Children Sum Property in Binary Tree | O\(N\) Approach | C++ | Java](#)



Question Link: [https://www.codingninjas.com/codestudio/problems/childrensumproperty\\_790723](https://www.codingninjas.com/codestudio/problems/childrensumproperty_790723)

source=youtube&campaign=Striver Tree Videos&utm\_source=youtube&utm\_medium=affiliate&utm\_campaign=Striver Tree Videos

NOTE/WARNING:

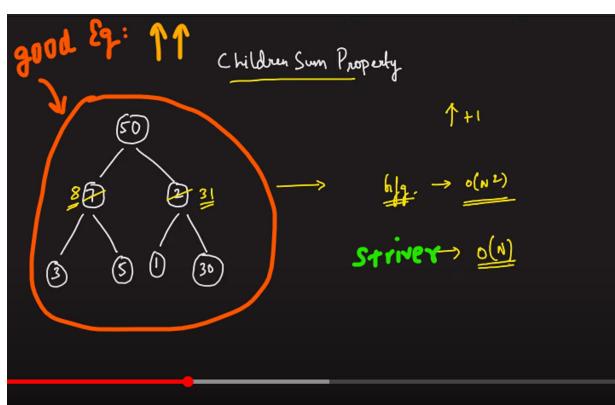
This question is NOT as EASY as it seems

**GFG solution: O(N\*N) ::** <https://www.geeksforgeeks.org/convert-an-arbitrary-binary-tree-to-a-tree-that-holds-children-sum-property/?ref=rp>  
**Striver's Solution: O(N) ::** [https://github.com/striver79/FreeKaTreeSeries/blob/main/L29\\_childrenSumProblemCpp](https://github.com/striver79/FreeKaTreeSeries/blob/main/L29_childrenSumProblemCpp)

Both solutions have their own pro's and cons.

Had the question been to make the tree satisfy Children Sum property and make the tree as minimal as possible :

- Gfg's solution would have been the solution



**Striver's Solution:**

L29. Children Sum Property in Binary Tree | O(N) Approach | C++ | Java

```

1 class Solution {
2     void changeTree(BinaryTreeNode < int > * root) {
3         if(root == NULL) return;
4         int child = 0;
5         if(root->left) {
6             child += root->left->data;
7         }
8         if(root->right) {
9             child += root->right->data;
10        }
11    }
12    if(child >= root->data) root->data = child;
13    else {
14        if(root->left) root->left->data = root->data;
15        else if(root->right) root->right->data = root->data;
16    }
17    reorder(root->left);
18    reorder(root->right);
19    int tot = 0;
20    if(root->left) tot += root->left->data;
21    if(root->right) tot+= root->right->data;
22    if(root->left or root->right) root->data = tot;
23 }
24 
```

```

1 class Solution {
2     public static void changeTree(BinaryTreeNode < Integer > root) {
3         if(root == null) return;
4         int child = 0;
5         if(root.left != null) {
6             child += root.left.data;
7         }
8         if(root.right != null) {
9             child += root.right.data;
10        }
11    }
12    if(child >= root.data) root.data = child;
13    else {
14        if(root.left != null) root.left.data = root.data;
15        else if(root.right != null) root.right.data = root.data;
16    }
17    changeTree(root.left);
18    changeTree(root.right);
19    int tot = 0;
20    if(root.left != null) tot += root.left.data;
21    if(root.right != null) tot+= root.right.data;
22    if(root.left == null || root.right == null) root.data = tot;
23 }
24 
```

```

1 void reorder(BinaryTreeNode < int > * root) {
2     if(root == NULL) return;
3     int child = 0;
4     if(root->left) {
5         child += root->left->data;
6     }
7     if(root->right) {
8         child += root->right->data;
9     }
10    if(child >= root->data) root->data = child;
11    else {
12        if(root->left) root->left->data = root->data;
13        else if(root->right) root->right->data = root->data;
14    }
15    reorder(root->left);
16    reorder(root->right);
17    int tot = 0;
18    if(root->left) tot += root->left->data;
19    if(root->right) tot+= root->right->data;
20    if(root->left or root->right) root->data = tot;
21 }
22 void changeTree(BinaryTreeNode < int > * root) {
23     reorder(root);
24 }
```

My Solution(same as striver | using his idea of backtracking):

codestudio  
POWERED BY CODING NINJAS

Practice Guided Path Interview Prep Challenges Knowledge Centre Community

Problem Submissions Solution New Discuss C++ (g++ 5.4)

### Children Sum Property

Difficulty: MEDIUM Contributed By Ayush Thakur

Avg. time to solve 36 min Success Rate 70%

**Problem Statement**

Given a binary tree of nodes 'N', you need to modify the value of its nodes, such that the tree holds the Children sum property.

A binary tree is said to follow the children sum property if, for every node of that tree, the value of that node is equal to the sum of the value(s) of all of its children nodes( left child and the right child).

**Note:**

1. You can only increment the value of the nodes, in other words, the modified value must be at least equal to the original value of that node.
2. You can not change the structure of the original binary tree.
3. A binary tree is a tree in which each node has at most two children.
4. You can assume the value can be 0 for a NULL node and there can also be an empty tree.

**Input Format:**

The first line contains a single integer 'T' representing the number of test cases.

The first and the only line of each test case will contain the values of the nodes of the tree in the level order form ( -1 for NULL node) Refer to the example for further clarification.

```

19 *****
20 void changeTree(BinaryTreeNode<int> * root) {
21     // Write your code here.
22     if(root==nullptr) return;
23     int sum=0;
24
25     BinaryTreeNode<int>* left=root->left;
26     BinaryTreeNode<int>* right=root->right;
27
28     if(left != nullptr) sum+=left->data;
29     if(right != nullptr) sum+=right->data;
30     if(sum > root->data) root->data=sum;
31     else if(sum < root->data){
32         if(left != nullptr) left->data= root->data;
33         else if(right != nullptr) right->data = root->data;
34     }
35     changeTree(left);
36     changeTree(right);
37
38     sum=0; //backtracking part
39     if(left != nullptr) sum+=left->data;
40     if(right != nullptr) sum+=right->data;
41     if(sum > root->data) root->data=sum;
42 }
```

Show Hint Last saved on 5:55:26 PM Run Code Submit

### L30. Print all the Nodes at a distance of K in Binary Tree | C++ | Java \*\*\*\*\*

## NOT DONE BY SELF :: DO AGAIN

Question Link : <https://leetcode.com/problems/all-nodes-distance-k-in-binary-tree/>

863. All Nodes Distance K in Binary Tree

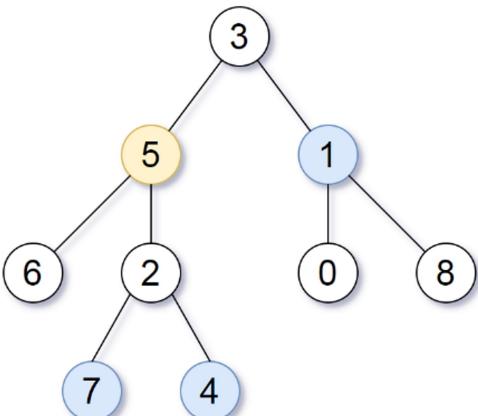
Medium 5057 99 Add to List Share

Given the `root` of a binary tree, the value of a target node `target`, and an integer `k`, return an array of the values of all nodes that have a distance `k` from the target node.

You can return the answer in **any order**.

**Example 1:**

**Example 1:**



Input: `root = [3,5,1,6,2,0,8,null,null,7,4]`, `target = 5`, `k = 2`

Output: `[7,4,1]`

Explanation: The nodes that are a distance 2 from the target node (with value 5) have

**Example 2:**

```
Input: root = [1], target = 1, k = 3
Output: []
```

**Constraints:**

- The number of nodes in the tree is in the range [1, 500].
- $0 \leq \text{Node.val} \leq 500$ .
- All the values `Node.val` are **unique**.
- `target` is the value of one of the nodes in the tree.
- $0 \leq k \leq 1000$

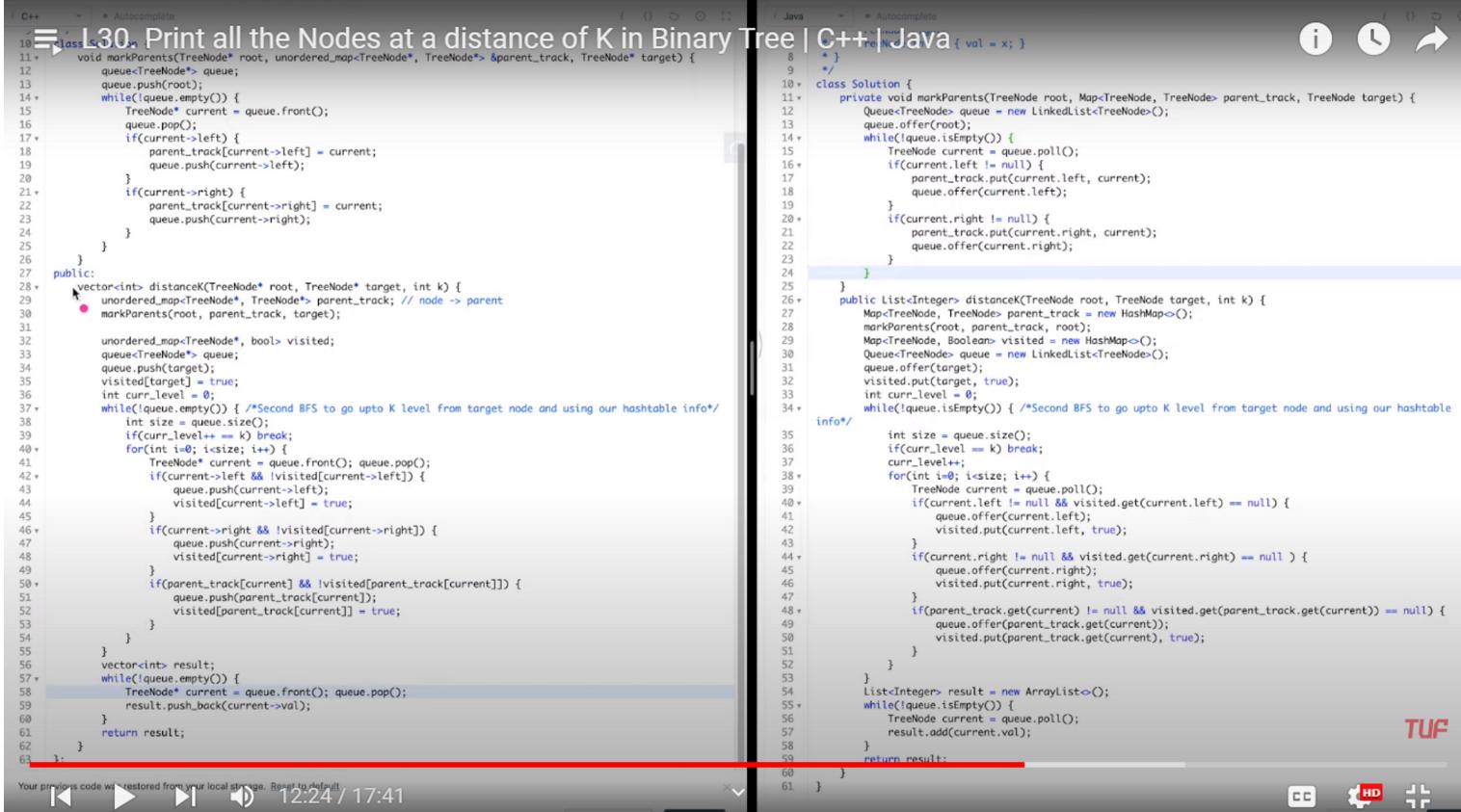
Accepted 186,562 | Submissions 311,208

Seen this question in a real interview before?

 Companies 

Related Topics

Tree Depth-First Search Breadth-First Search Binary Tree

**Striver Solution:**


```
C++ // Autocomplete
10 class Solution {
11     void markParents(TreeNode* root, unordered_map<TreeNode*, TreeNode*> &parent_track, TreeNode* target) {
12         queue<TreeNode*> queue;
13         queue.push(root);
14         while(!queue.empty()) {
15             TreeNode* current = queue.front();
16             queue.pop();
17             if(current->left) {
18                 parent_track[current->left] = current;
19                 queue.push(current->left);
20             }
21             if(current->right) {
22                 parent_track[current->right] = current;
23                 queue.push(current->right);
24             }
25         }
26     }
27     public:
28     vector<int> distanceK(TreeNode* root, TreeNode* target, int k) {
29         unordered_map<TreeNode*, TreeNode*> parent_track; // node -> parent
30         markParents(root, parent_track, target);
31         unordered_map<TreeNode*, bool> visited;
32         queue<TreeNode*> queue;
33         queue.push(target);
34         visited[target] = true;
35         int curr_level = 0;
36         while(!queue.empty()) { /*Second BFS to go upto K level from target node and using our hashtable info*/
37             int size = queue.size();
38             if(curr_level++ == k) break;
39             for(int i=0; i<size; i++) {
40                 TreeNode* current = queue.front(); queue.pop();
41                 if(current->left && !visited[current->left]) {
42                     queue.push(current->left);
43                     visited[current->left] = true;
44                 }
45                 if(current->right && !visited[current->right]) {
46                     queue.push(current->right);
47                     visited[current->right] = true;
48                 }
49                 if(parent_track[current] && !visited[parent_track[current]]) {
50                     queue.push(parent_track[current]);
51                     visited[parent_track[current]] = true;
52                 }
53             }
54         }
55         vector<int> result;
56         while(!queue.empty()) {
57             TreeNode* current = queue.front(); queue.pop();
58             result.push_back(current->val);
59         }
60     }
61     return result;
62 }
63 
```

```
Java // Autocomplete
8 * */
9 */
10 class Solution {
11     private void markParents(TreeNode root, Map<TreeNode, TreeNode> parent_track, TreeNode target) {
12         Queue<TreeNode> queue = new LinkedList<TreeNode>();
13         queue.offer(root);
14         while(!queue.isEmpty()) {
15             TreeNode current = queue.poll();
16             if(current.left != null) {
17                 parent_track.put(current.left, current);
18                 queue.offer(current.left);
19             }
20             if(current.right != null) {
21                 parent_track.put(current.right, current);
22                 queue.offer(current.right);
23             }
24         }
25     }
26     public List<Integer> distanceK(TreeNode root, TreeNode target, int k) {
27         Map<TreeNode, TreeNode> parent_track = new HashMap<>();
28         markParents(root, parent_track, root);
29         Map<TreeNode, Boolean> visited = new HashMap<>();
30         Queue<TreeNode> queue = new LinkedList<TreeNode>();
31         queue.offer(target);
32         visited.put(target, true);
33         int curr_level = 0;
34         while(!queue.isEmpty()) { /*Second BFS to go upto K level from target node and using our hashtable
info*/
35             int size = queue.size();
36             if(curr_level++ == k) break;
37             curr_level++;
38             for(int i=0; i<size; i++) {
39                 TreeNode current = queue.poll();
40                 if(current.left != null && visited.get(current.left) == null) {
41                     queue.offer(current.left);
42                     visited.put(current.left, true);
43                 }
44                 if(current.right != null && visited.get(current.right) == null ) {
45                     queue.offer(current.right);
46                     visited.put(current.right, true);
47                 }
48                 if(parent_track.get(current) != null && visited.get(parent_track.get(current)) == null) {
49                     queue.offer(parent_track.get(current));
50                     visited.put(parent_track.get(current), true);
51                 }
52             }
53         }
54         List<Integer> result = new ArrayList<>();
55         while(!queue.isEmpty()) {
56             TreeNode current = queue.poll();
57             result.add(current.val);
58         }
59     }
60     return result;
61 }
```

Your previous code will be restored from your local storage. Reset to default

12:24 / 17:41

**My method(incomplete):**

```
class Solution {
public:
    vector<int> distanceK(TreeNode* root, TreeNode* target, int k) {
        vector<int> ans,v;
        //nodes at distance k below target
        belowA(target,target,k,v);
        ans.insert(ans.end(),v.begin(),v.end());
        v.clear();
        //nodes at distance k above target
        //1st get depth of target node
        vector<pair<TreeNode*,int>> path;// {root to target nodes , level}

        int depTarget=0;
        depth(root,target,depTarget);
```

```
    return ans;
}
void belowA(TreeNode* root, TreeNode* target, int a, vector<int> &v){
    //any traversal should work, just that root!=target & level a from root
    if(a==0 && target!=root){
        //a=dpath left to go
        if(root!=nullptr) v.push_back(root->val);
        return;
    }
    belowA(root->left,target,a-1,v);
    belowA(root->right,target,a-1,v);
}
void depth(TreeNode* root,TreeNode* target,int& d){
    if(root==target) return d;
    depth(root->left,target,d+1);
    depth(root->right,target,d+1);
}
vector<pair<TreeNode*,int>> pathRootToTarget(TreeNode* root,TreeNode* target,int d,vector<pair<TreeNode*,int>> path){
    if(root==target) return path;
    if(root==nullptr)
    }
};

=====
```

L31.

L32.

L33.

L34.

L35.

L36.

L37.

L38.

L39.

L40.

L41.

L42.

L43.