Trees TUF

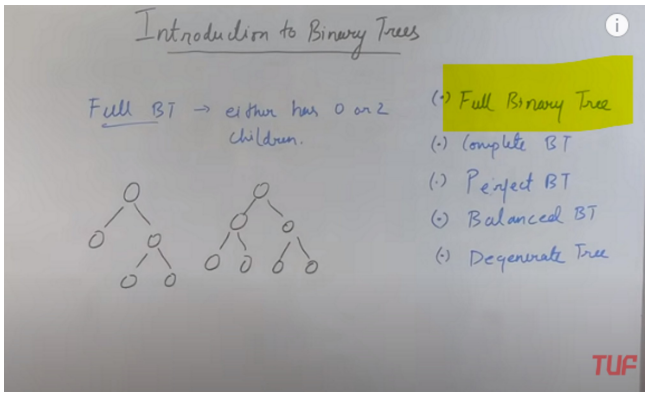
25 October 2021 18:24

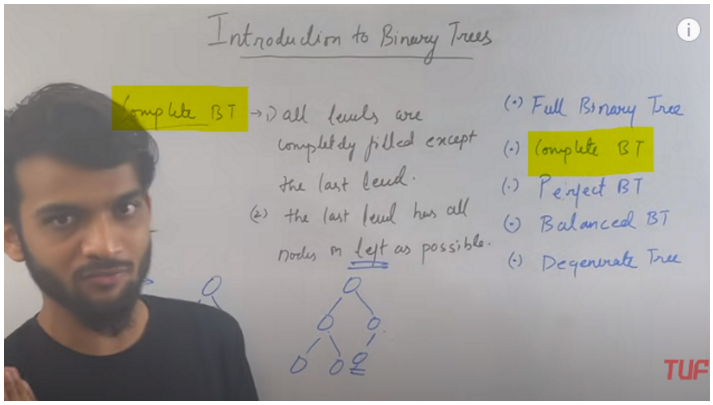
https://techinterviewhandbook.org/algorithms/tree/

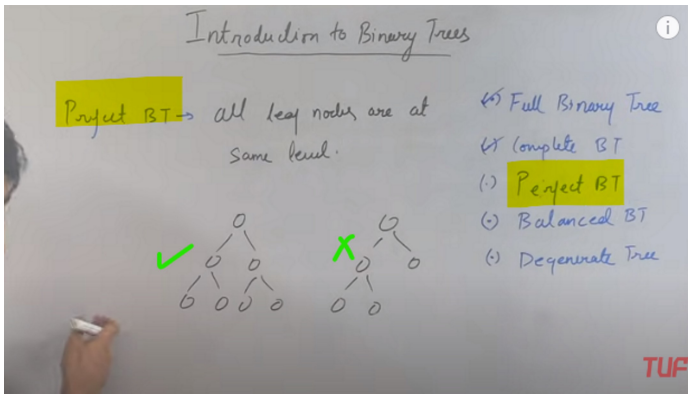
https://github.com/striver79/FreeKaTreeSeries

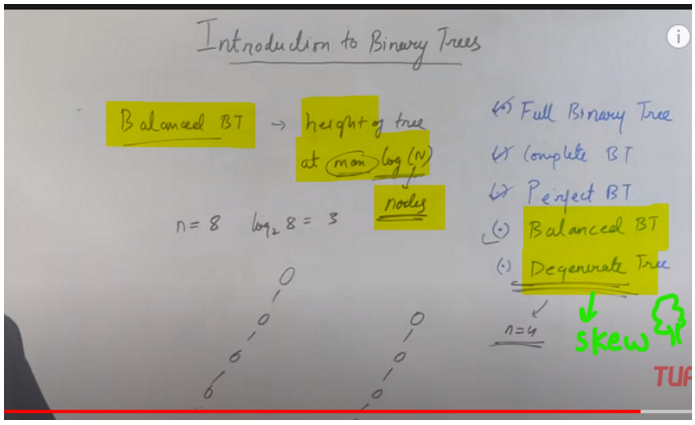
** => nice for understanding

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







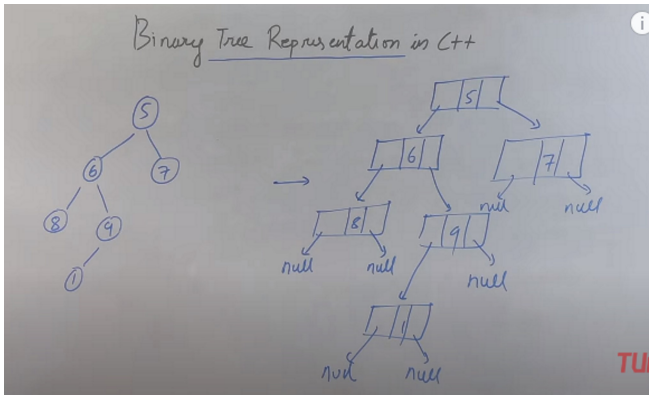


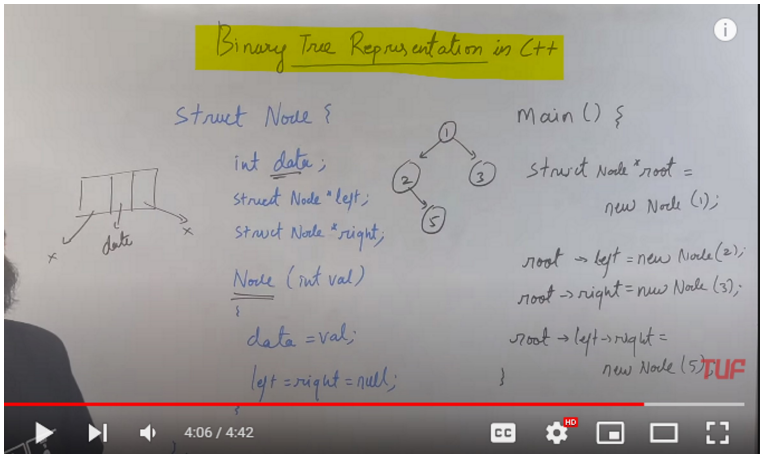
Balanced binary tree

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**. ... 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):

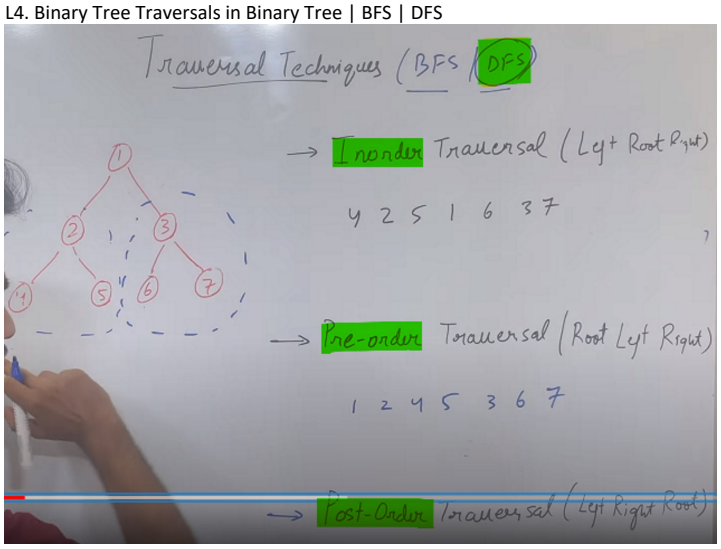
```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode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

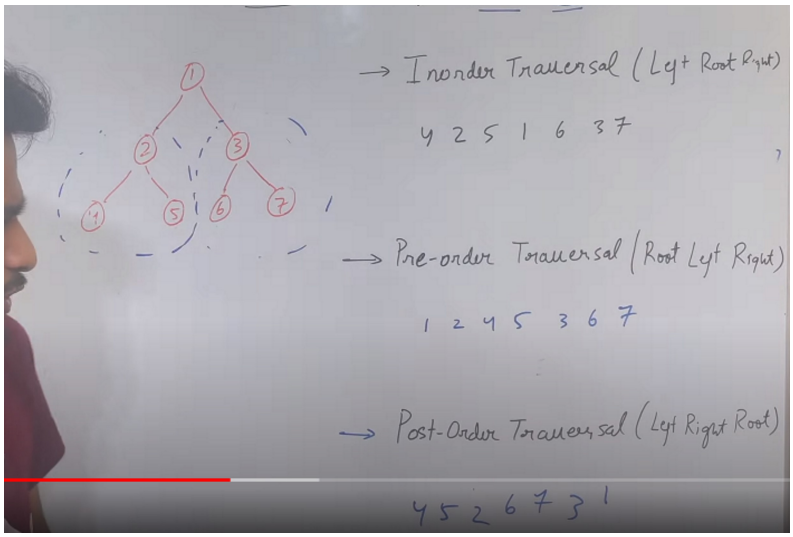
****Note** :** Using nullptr is better than null for left and right tree pointers , as it prevents errors in many cases (and there is a logical explanation for that, because of data types)

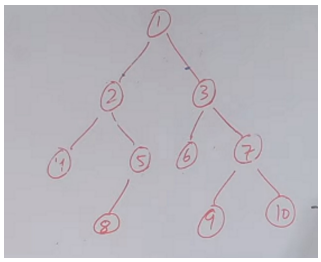
Null => in c++ can mean both as a integer and as null refernce ,and so it depends on the usage
But nullptr : explicitly means that it is a pointer type, so its better to use it in case of pointer

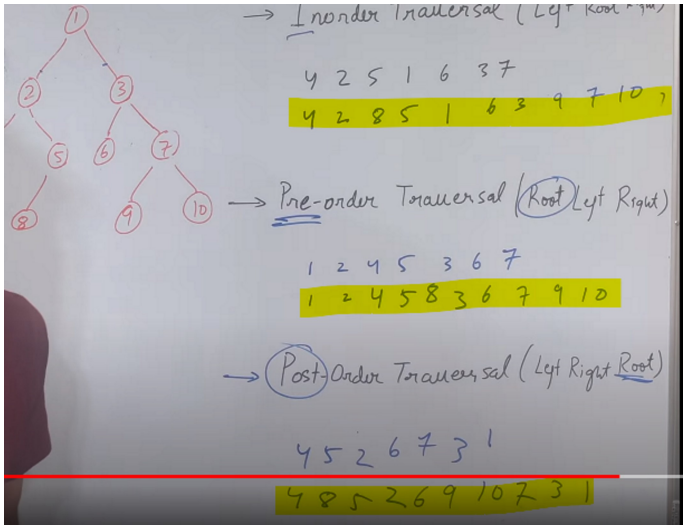
==========

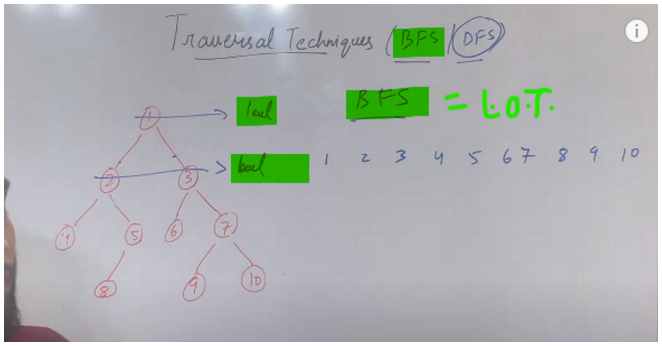
L4. Binary Tree Traversals in Binary Tree | BFS | DFS

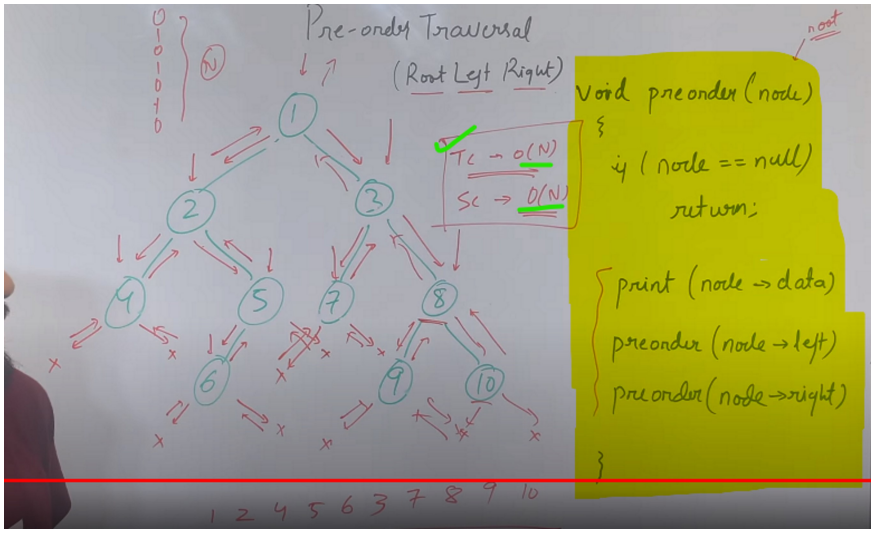












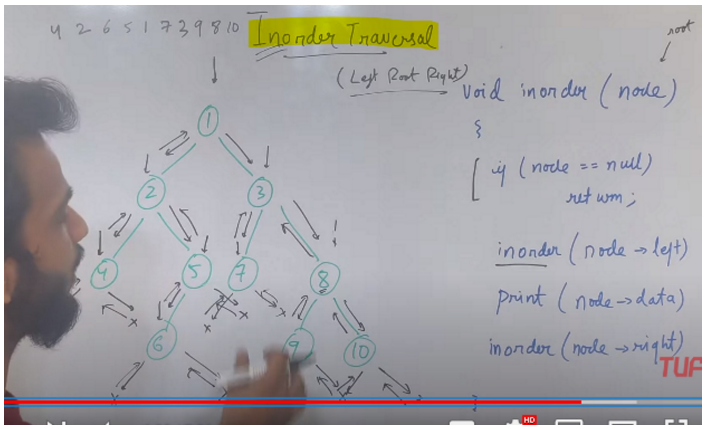
// LC link : https://leetcode.com/problems/binary-tree-preorder-traversal

```
/**
 * 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;
```

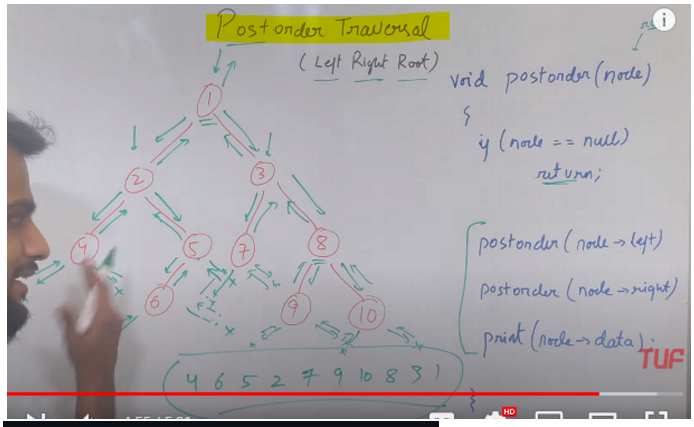
```
        preorder.push_back(node->val);
        dfs(node->left, preorder);
        dfs(node->right, preorder);
    }
public:
    vector<int> preorderTraversal(TreeNode* root) {
        vector<int> preorder;
        dfs(root, preorder);
        return preorder;
    }
};
```


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



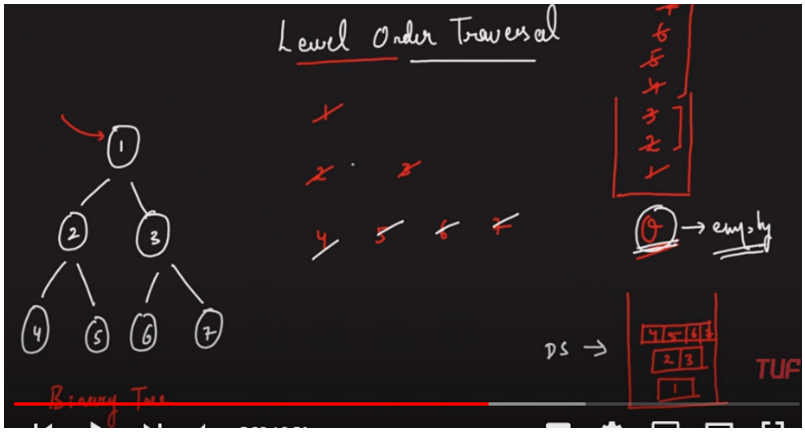

```
class Solution {
private:
    void dfs(TreeNode *node, vector<int> &inorder) {
        if(node == NULL) return;

        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;
    }
};
```

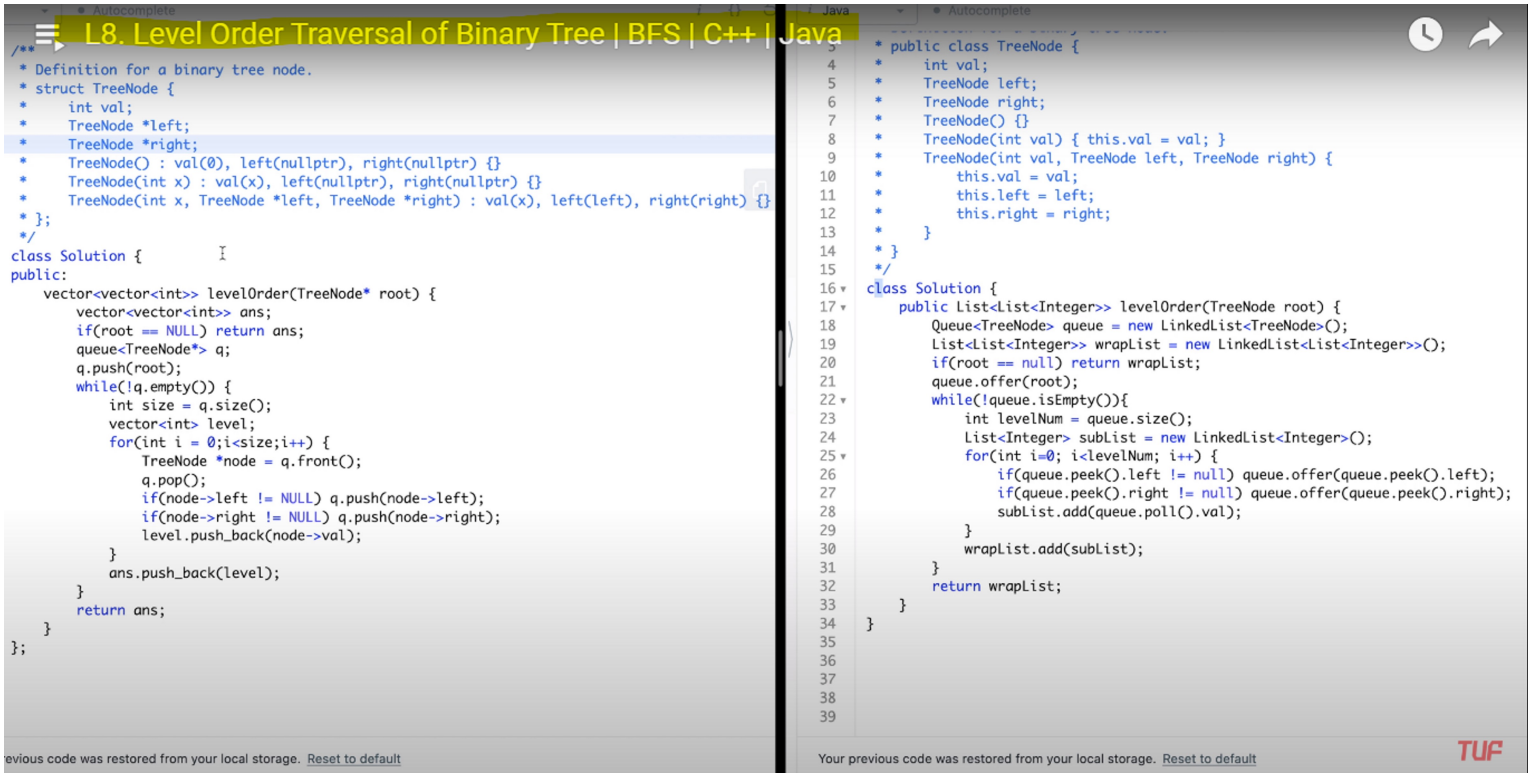



```
class Solution {
private:
```

```
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>

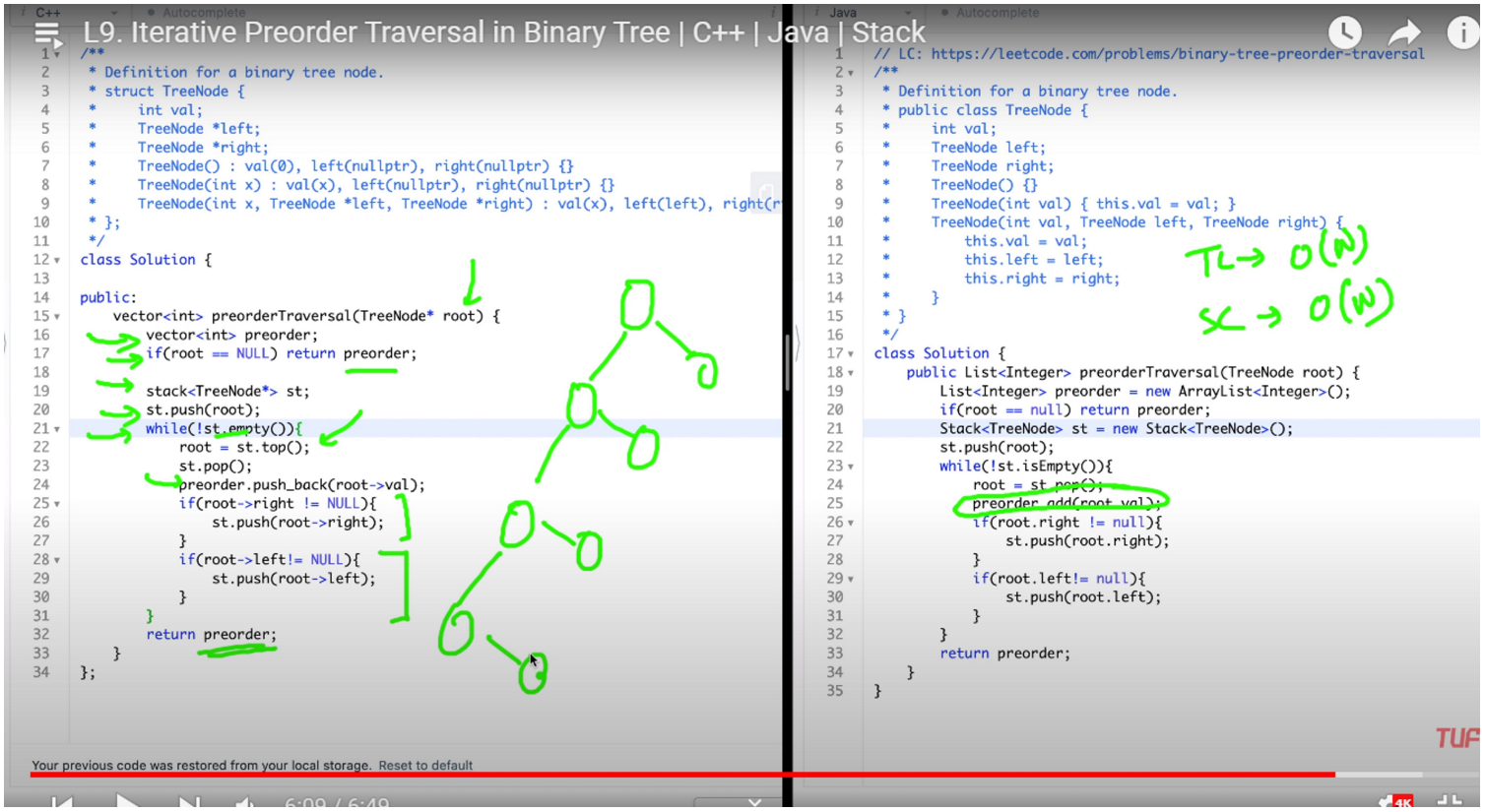




==========

L9. Iterative Preorder Traversal

TC= O(n) , SC = O(n) worst case tree shown in green below)




```cpp
/**
 * 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:
    //approach 1: recursive
    //approach 2: iterative (curremt)
    vector<int> preorderTraversal(TreeNode* root) {
        vector<int> ans;
        if(root==nullptr) return ans;

        stack<TreeNode*> s;
        s.push(root);
        while(!s.empty()){
            ans.push_back(s.top()->val);
            TreeNode* temp=s.top();
            s.pop();
            if(temp->right!=nullptr) s.push(temp->right);
            if(temp->left!=nullptr) s.push(temp->left);
        }
        return ans;
    }
};
```

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)

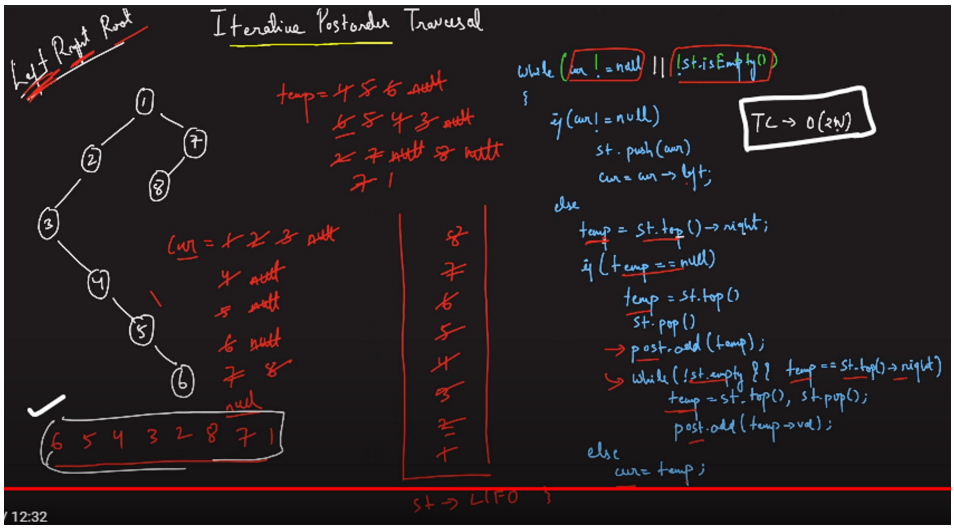
```cpp
class Solution {

public:
    vector<int> inorderTraversal(TreeNode* root) {
        stack<TreeNode*> st;
        TreeNode* node = root;
        vector<int> inorder;
        while(true) {
            if(node != NULL) {
                st.push(node);
                node = node->left;
            }
            else {

                if(st.empty() == true) break;
                node = st.top();
                st.pop();
                inorder.push_back(node->val);
                node = node->right;
            }
        }
        return inorder;
    }
};
```

==========

L11: Iterative Postorder traversal with 2 stacks
TC=O(N) , SC= O(N)

==========

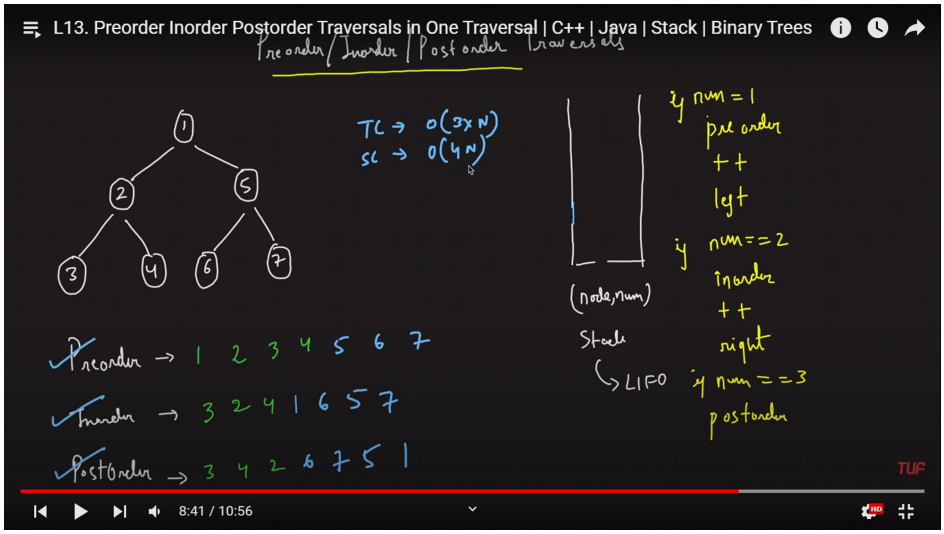
L12: Iterative Postorder traversal with single stack
TC=O(2*N) ,Auxillary SC= O(N) (used by stack)

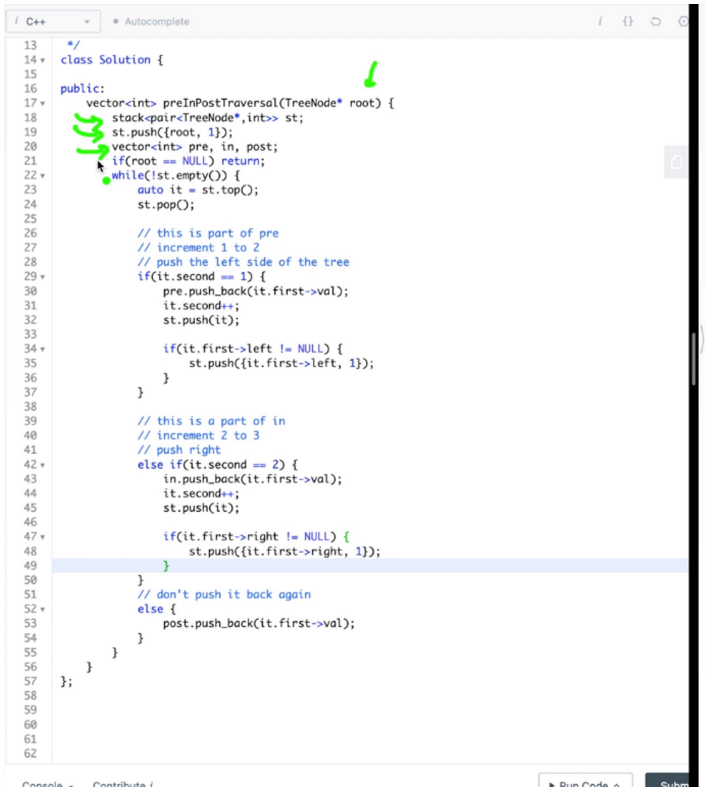


==========

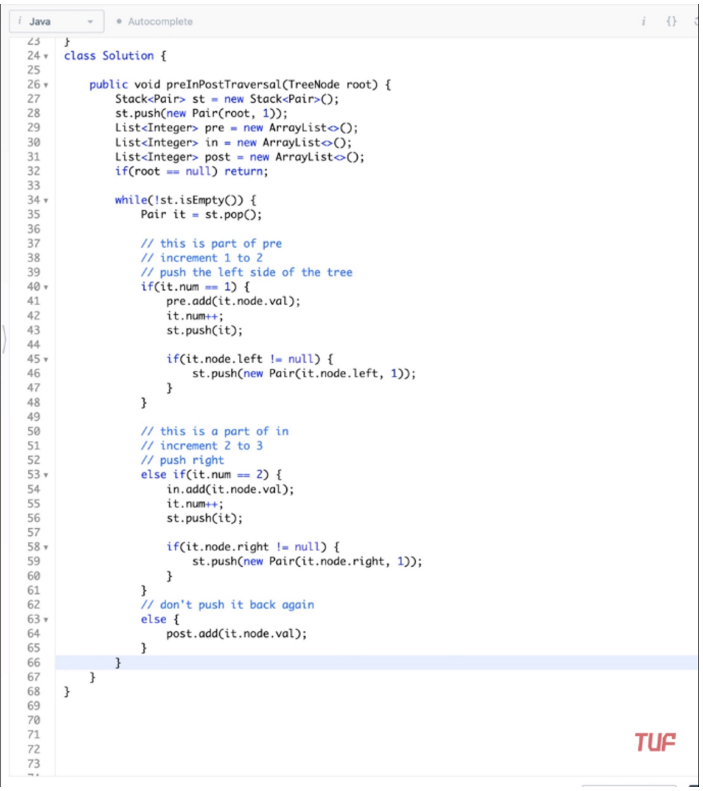
L13: Pre + In + Post : Order ,All 3 traversals in just one go(/traversal)

- **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**





```cpp
*/
class Solution {

public:
    vector<int> preInPostTraversal(TreeNode* root) {
        stack<pair<TreeNode*,int>> st;
        st.push({root, 1});
        vector<int> pre, in, post;
        if(root == NULL) return;
        while(!st.empty()) {
            auto it = st.top();
            st.pop();

            // this is part of pre
            // increment 1 to 2
            // push the left side of the tree
            if(it.second == 1) {
                pre.push_back(it.first->val);
                it.second++;
                st.push(it);

                if(it.first->left != NULL) {
                    st.push({it.first->left, 1});
                }
            }

            // this is a part of in
            // increment 2 to 3
            // push right
            else if(it.second == 2) {
                in.push_back(it.first->val);
                it.second++;
                st.push(it);

                if(it.first->right != NULL) {
                    st.push({it.first->right, 1});
                }
            }
            // don't push it back again
            else {
                post.push_back(it.first->val);
            }
        }
    }
};
```

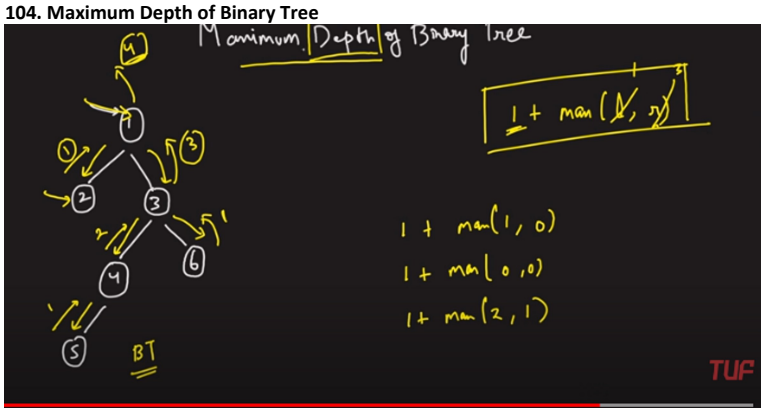


```java
}
class Solution {

    public void preInPostTraversal(TreeNode root) {
        Stack<Pair> st = new Stack<Pair>();
        st.push(new Pair(root, 1));
        List<Integer> pre = new ArrayList<>();
        List<Integer> in = new ArrayList<>();
        List<Integer> post = new ArrayList<>();
        if(root == null) return;

        while(!st.isEmpty()) {
            Pair it = st.pop();

            // this is part of pre
            // increment 1 to 2
            // push the left side of the tree
            if(it.num == 1) {
                pre.add(it.node.val);
                it.num++;
                st.push(it);

                if(it.node.left != null) {
                    st.push(new Pair(it.node.left, 1));
                }
            }

            // this is a part of in
            // increment 2 to 3
            // push right
            else if(it.num == 2) {
                in.add(it.node.val);
                it.num++;
                st.push(it);

                if(it.node.right != null) {
                    st.push(new Pair(it.node.right, 1));
                }
            }
            // don't push it back again
            else {
                post.add(it.node.val);
            }
        }
    }
}
```

==========

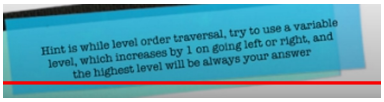
Lec 14 ||

104. Maximum Depth of Binary Tree



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

1. **Using recursion**
2. **Using level order traversal**

a. 

Soln 1: Recursive:

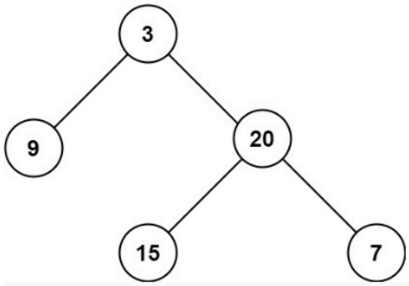
104. Maximum Depth of Binary Tree

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]
```

```
/**
 * 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:
    int maxDepth(TreeNode* root) {
        if(root==nullptr) return 0;
        return 1+max(maxDepth(root->left),maxDepth(root->right));

    }
};
```

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 — October LeetCoding Challenge 2021 — Premium

Description | Solution | Discuss (999+) | Submissions

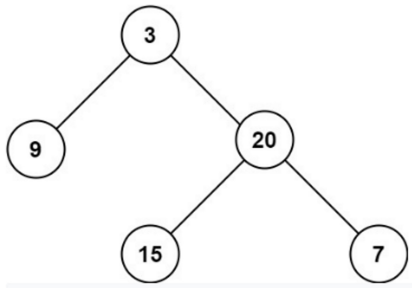
104. Maximum Depth of Binary Tree

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]
```

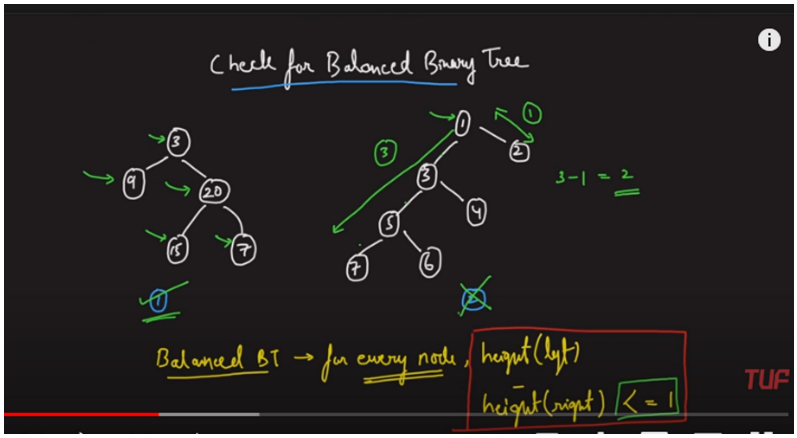
C++ Autocomplete

```
/**
 * 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), righ
 * };
 */
class Solution {
public:
    //approach 1: recursion (this is better)
    //approach 2: by level order traversal [current]

    int maxDepth(TreeNode* root) {
        //level order traversal
        int level=0;
        if(root==nullptr) return 0;
        queue<TreeNode*> q;
        q.push(root);
        while(!q.empty()){
            int n=q.size();
            for(int i=0;i<n;i++){
                if(q.front()->left != nullptr) q.push(q.front()->left);
                if(q.front()->right != nullptr) q.push(q.front()->right);
                q.pop();
            }
            level++;
        }
        return level;
    }
};
```

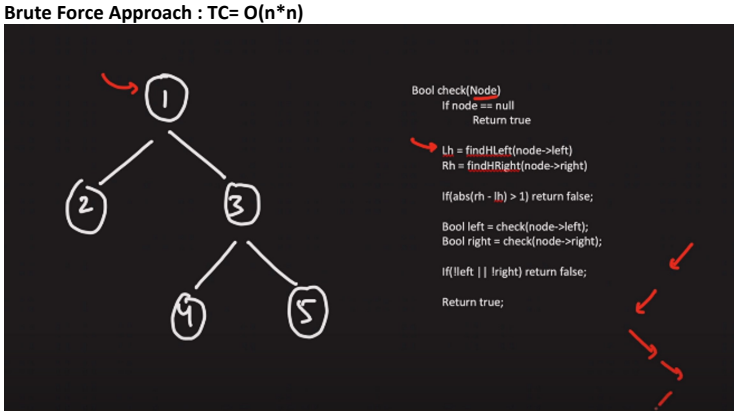
==========

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

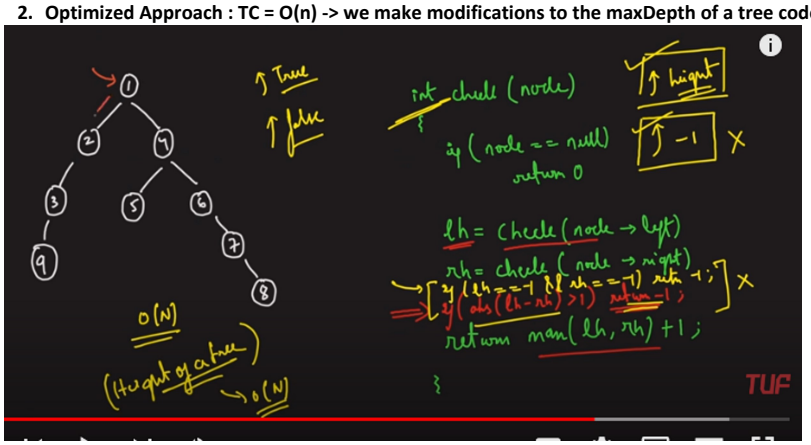
https://leetcode.com/problems/balanced-binary-tree/



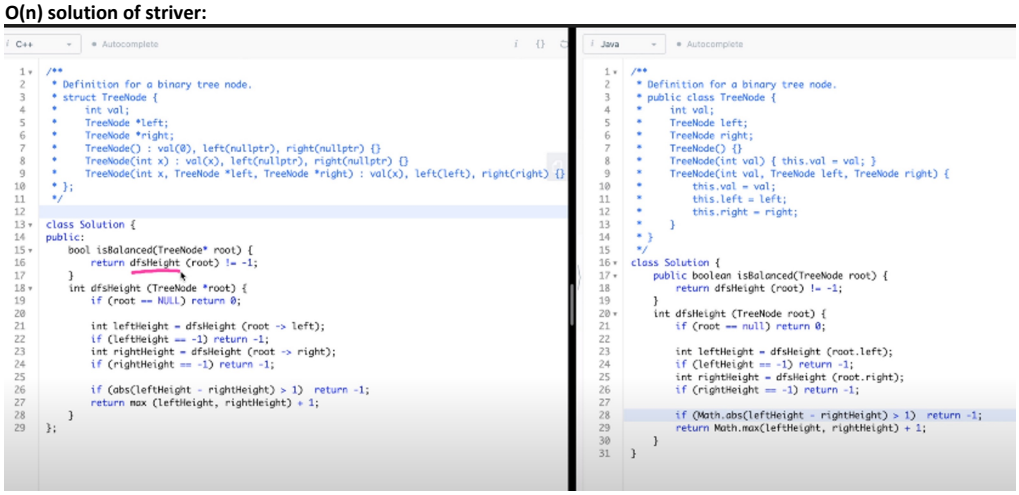
1. **Brute Force Approach : TC= O(n*n)**



2. **Optimized Approach : TC = O(n) -> we make modifications to the maxDepth of a tree code**



O(n) solution of striver:



```cpp
/**
 * 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:
    bool isBalanced(TreeNode* root) {
        return dfsHeight (root) != -1;
    }
    int dfsHeight (TreeNode *root) {
        if (root == NULL) return 0;

        int leftHeight = dfsHeight (root -> left);
        if (leftHeight == -1) return -1;
        int rightHeight = dfsHeight (root -> right);
        if (rightHeight == -1) return -1;

        if (abs(leftHeight - rightHeight) > 1)  return -1;
        return max (leftHeight, rightHeight) + 1;
    }
};
```

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public boolean isBalanced(TreeNode root) {
        return dfsHeight (root) != -1;
    }
    int dfsHeight (TreeNode root) {
        if (root == null) return 0;

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

        if (Math.abs(leftHeight - rightHeight) > 1)  return -1;
        return Math.max(leftHeight, rightHeight) + 1;
    }
}
```

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

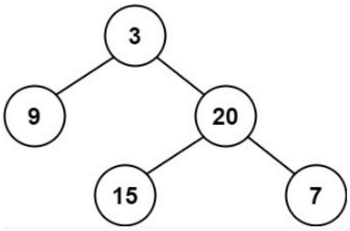
Easy 4505 258 Add to List Share

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:




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

C++ | Autocomplete

```
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    int maxDepth(TreeNode* node,bool &ans){
        if(node==nullptr || 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

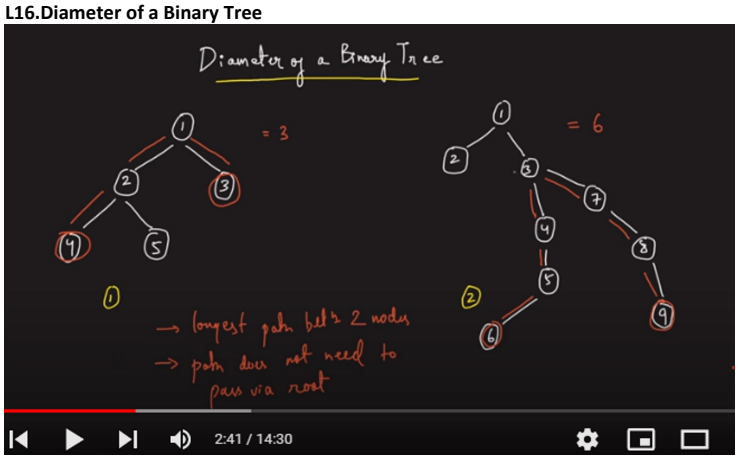
Your input `[1,2,2,3,3,null,null,4,4]`

Output `false`

Expected `false`

==========

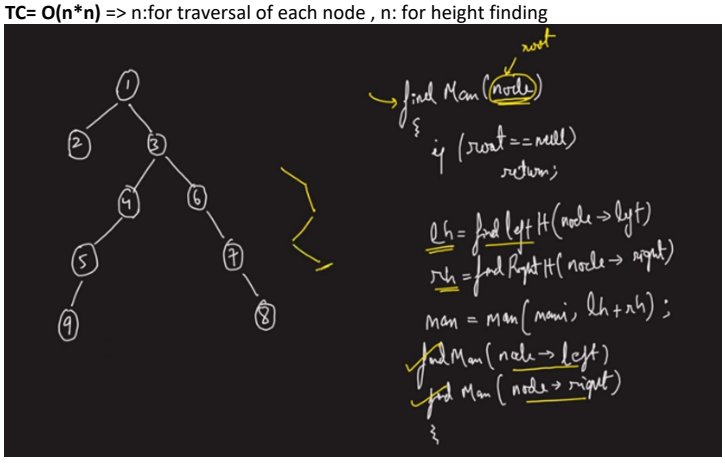
L16.Diameter of a Binary Tree



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 ans= max(left_height+right_height,ans) , for each node
TC= O(n*n) => n:for traversal of each node , n: for height finding



Description | Solution | Discuss (999+) | Submissions

543. Diameter of Binary Tree

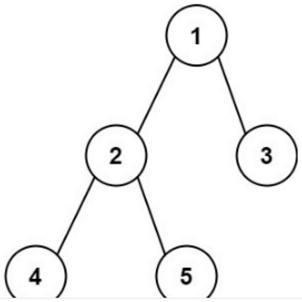
Easy 6272 388 Add to List Share

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:

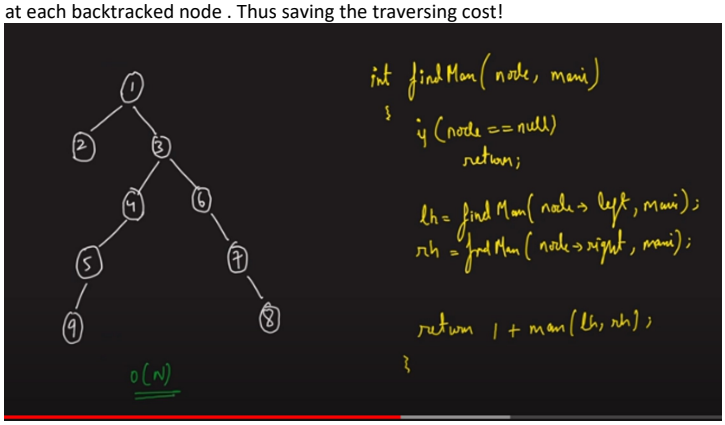


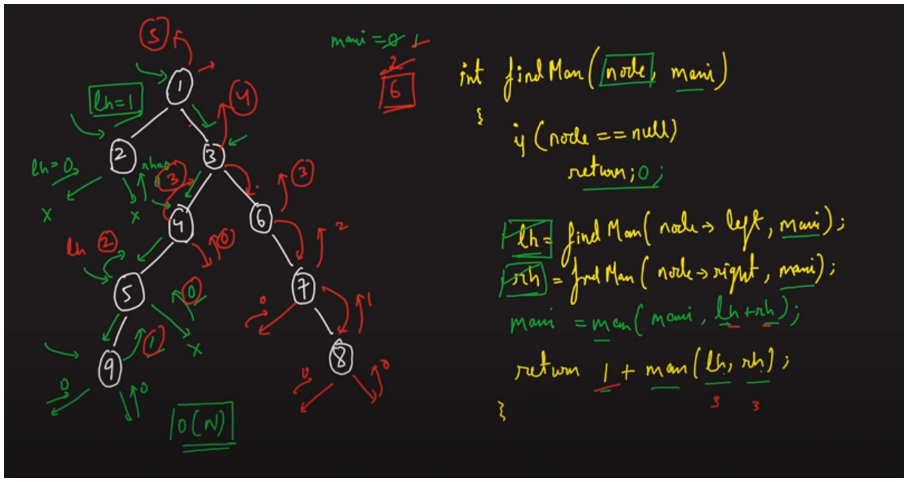
C++ | Autocomplete

```cpp
/**
 * 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:
    int maxDepth(TreeNode* node){
        if(node==nullptr) return 0;
        return 1+max(maxDepth(node->left),maxDepth(node->right));
    }

    void traverse(TreeNode* node,int &ans){
        if(node==nullptr) return;

        int lh=maxDepth(node->left);
        int rh=maxDepth(node->right);
        ans=max(ans,lh+rh);
        traverse(node->left,ans);
        traverse(node->right,ans);
    }

    int diameterOfBinaryTree(TreeNode* root) {
        int ans=0,lh,rh;
        if(root==nullptr) return ans;

        traverse(root,ans);
        return ans;
    }
};
```

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 refernce) at each backtracked node . Thus saving the traversing cost!





```cpp
/**
 * 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:
    int diameterOfBinaryTree(TreeNode* root) {
        int diameter = 0;
        height(root, diameter);
        return diameter;
    }
private:
    int height(TreeNode* node, int& diameter) {
        if (!node) {
            return 0;
        }
        int lh = height(node->left, diameter);
        int rh = height(node->right, diameter);
        diameter = max(diameter, lh + rh);
        return 1 + max(lh, rh);
    }
};
```

TC → O(N)

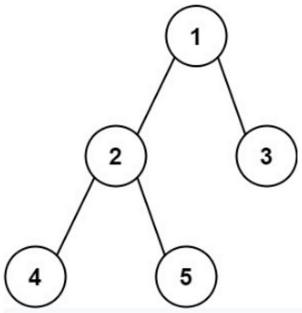
SC → O(N)

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
public class Solution {
    public int diameterOfBinaryTree(TreeNode root) {
        int[] diameter = new int[1];
        height(root, diameter);
        return diameter[0];
    }

    private int height(TreeNode node, int[] diameter) {
        if (node == null) {
            return 0;
        }
        int lh = height(node.left, diameter);
        int rh = height(node.right, diameter);
        diameter[0] = Math.max(diameter[0], lh + rh);
        return 1 + Math.max(lh, rh);
    }
}
```

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
```

```cpp
//approach 2: optimised
class Solution {
public:
    int diameterOfBinaryTree(TreeNode* root) {
        int diameter=0;
        if(root==nullptr) return 0;
        height(root,diameter); // int x= ....
        return diameter;
    }
    int height(TreeNode* node,int &diameter){
        if(node==nullptr) return 0;

        int lh=height(node->left,diameter);
        int rh=height(node->right,diameter);
        diameter=max(diameter,lh+rh);

        return 1+max(lh,rh);
    }
};
/*
// approach 1: brute force
class Solution {
public:
```

Testcase | Run Code Result | Debugger

Accepted Runtime: 0 ms

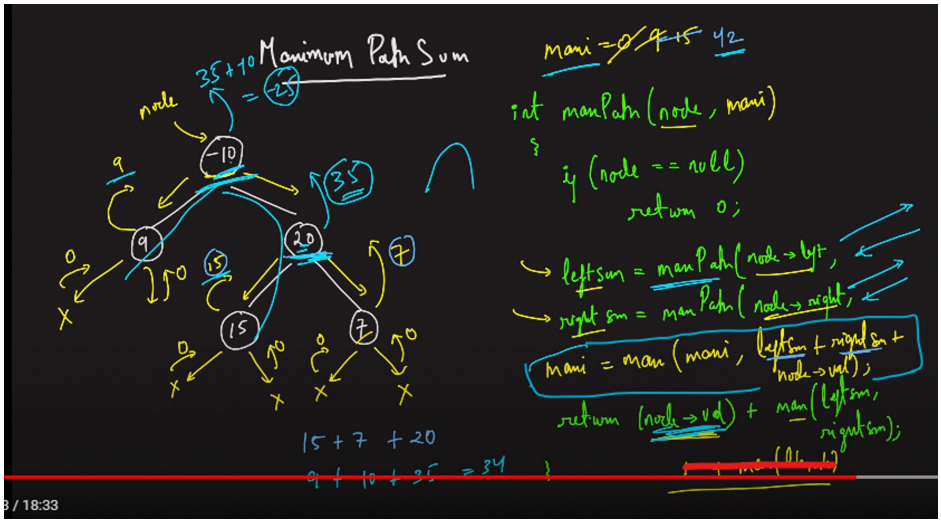
Your input [1,2,3,4,5]

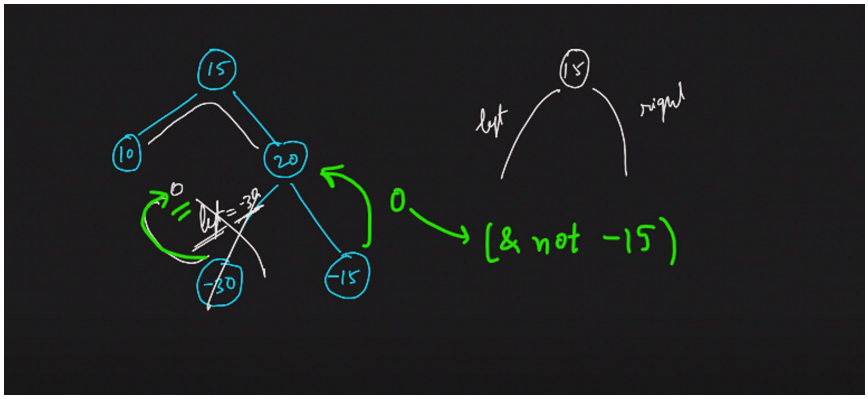
Output 3

Expected 3

==========

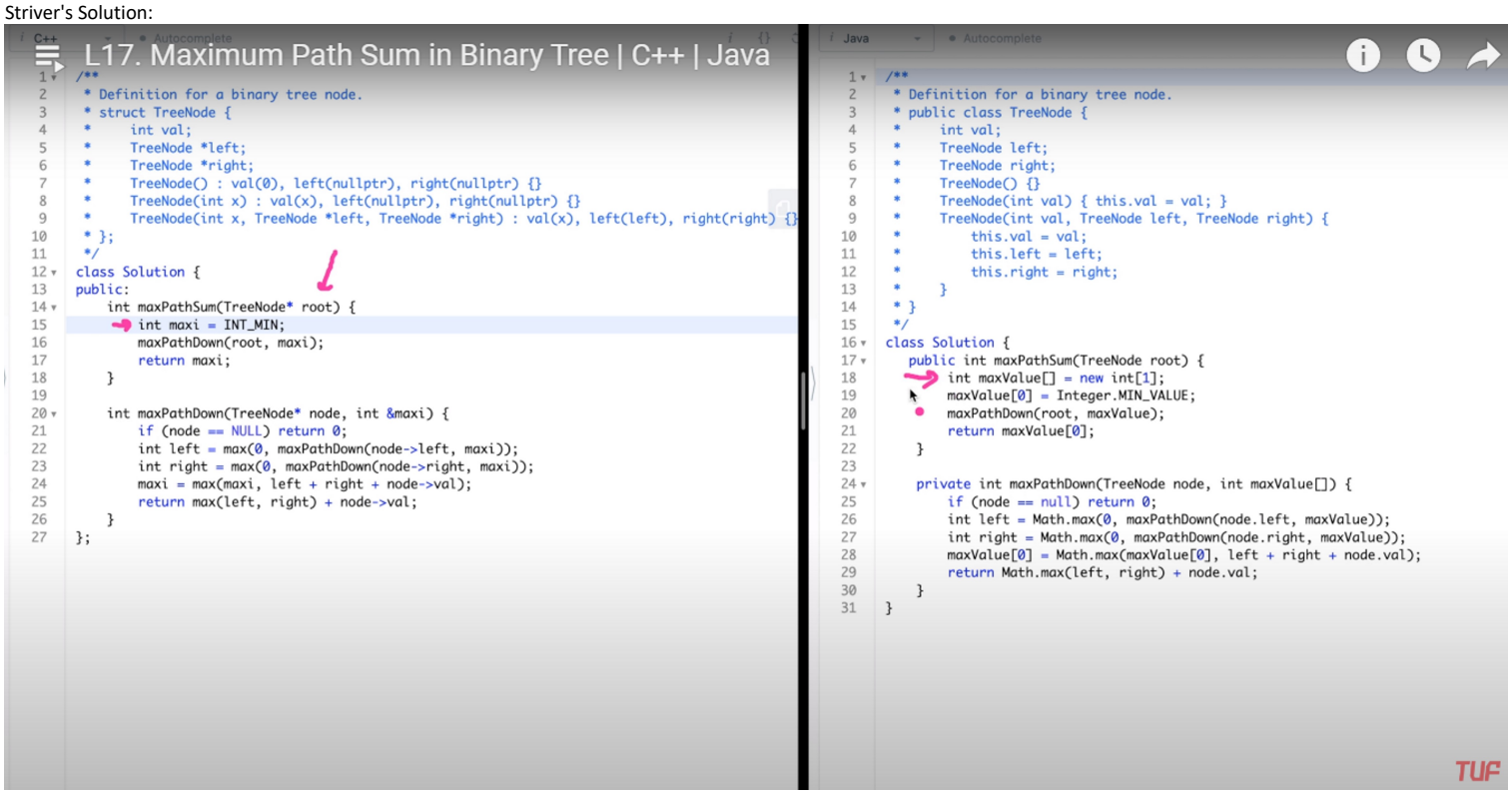
L17. 124 Binary Tree Max Path Sum





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

Striver's Solution:



My Solution:

Description | Solution | Discuss (999+) | Submissions

124. Binary Tree Maximum Path Sum

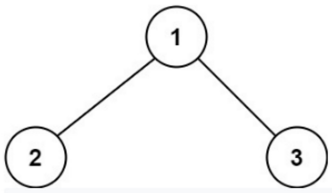
Hard | 7592 | 461 | Add to List | Share

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:




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

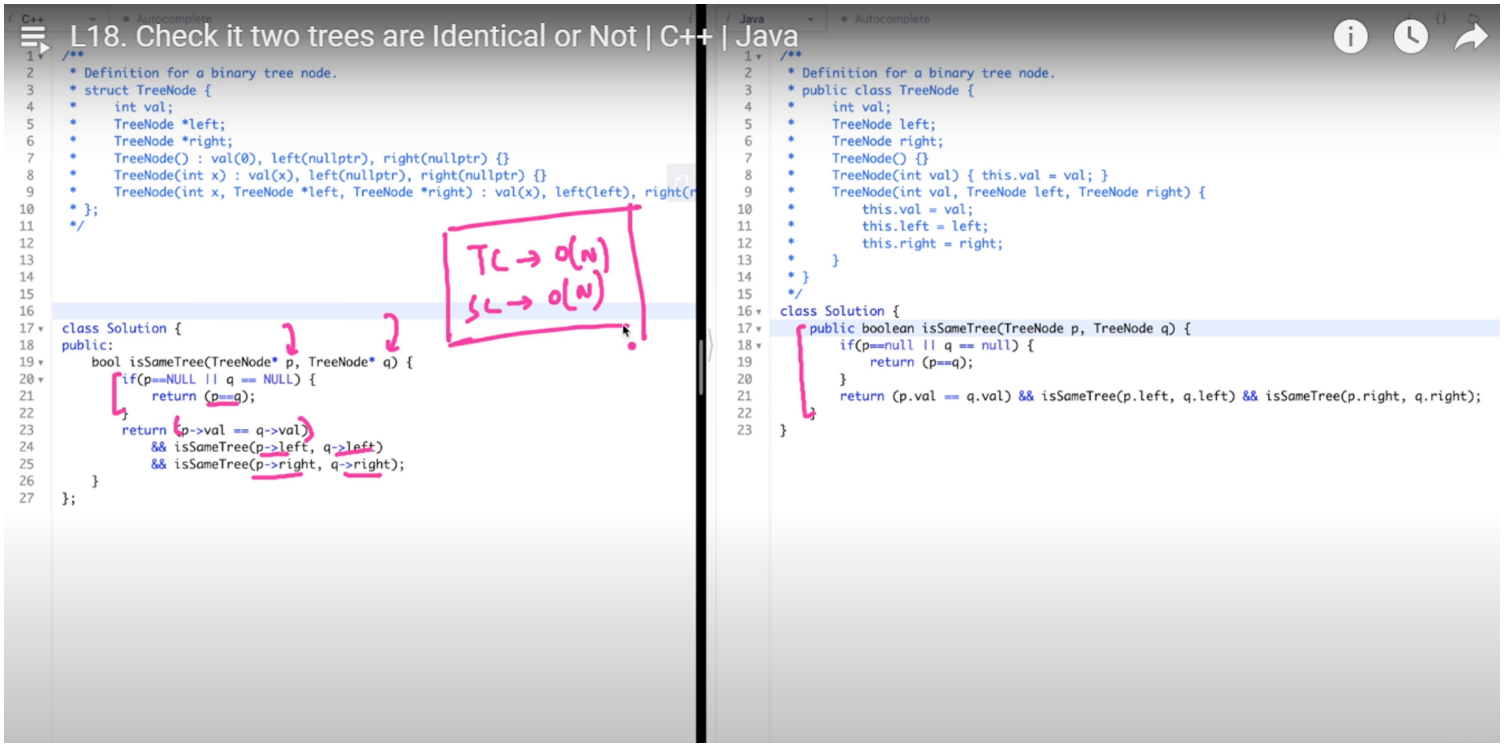
C++ | Autocomplete

```cpp
 *     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:
    int maxPathSum(TreeNode* root) {
        // if(root->)
        int sum=0,ans=INT_MIN,i;
        sum=pathSum(root,ans);
        // ans= ans<0 ? 0 : ans;
        return ans;
    }
    int pathSum(TreeNode* node,int& ans){
        if(node==nullptr) return 0;

        int maxLeft = pathSum(node->left,ans);
        int maxRight = pathSum(node->right,ans);
        //sum is the maximum path sum from current node and either its left subtree / its right subtree
        //(note its a straightline sum for its ancestor node's v-path sum )
        int sum = max(maxRight+node->val,maxLeft+node->val);
        sum = max(sum,node->val);

        //ans checks  & updates: if the current node's v-path sum is the max path sum
        ans = max(ans, node->val + maxLeft + maxRight);
        ans = max(ans, max(node->val + maxLeft, node->val + maxRight));
        ans = max(ans, node->val);
        // cout<<ans<<" ";

        return sum;
    }
};
```

Console | Contribute | Run Code | Submit

==========

L18.Same Tree or not




```cpp
/**
 * 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(r
 * };
 */

class Solution {
public:
    bool isSameTree(TreeNode* p, TreeNode* q) {
        if(p==NULL || q == NULL) {
            return (p==q);
        }
        return (p->val == q->val)
            && isSameTree(p->left, q->left)
            && isSameTree(p->right, q->right);
    }
};
```

Java | Autocomplete

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public boolean isSameTree(TreeNode p, TreeNode q) {
        if(p==null || q == null) {
            return (p==q);
        }
        return (p.val == q.val) && isSameTree(p.left, q.left) && isSameTree(p.right, q.right);
    }
}
```

LeetCode Explore Problems Interview Contest Discuss Store

LeetCoding Challenge + GIVEAWAY! Premium

Description Solution Discuss (999+) Submissions

C++ Autocomplete

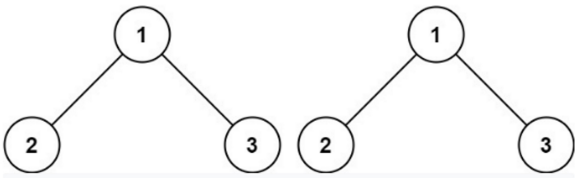
100. Same Tree

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
```

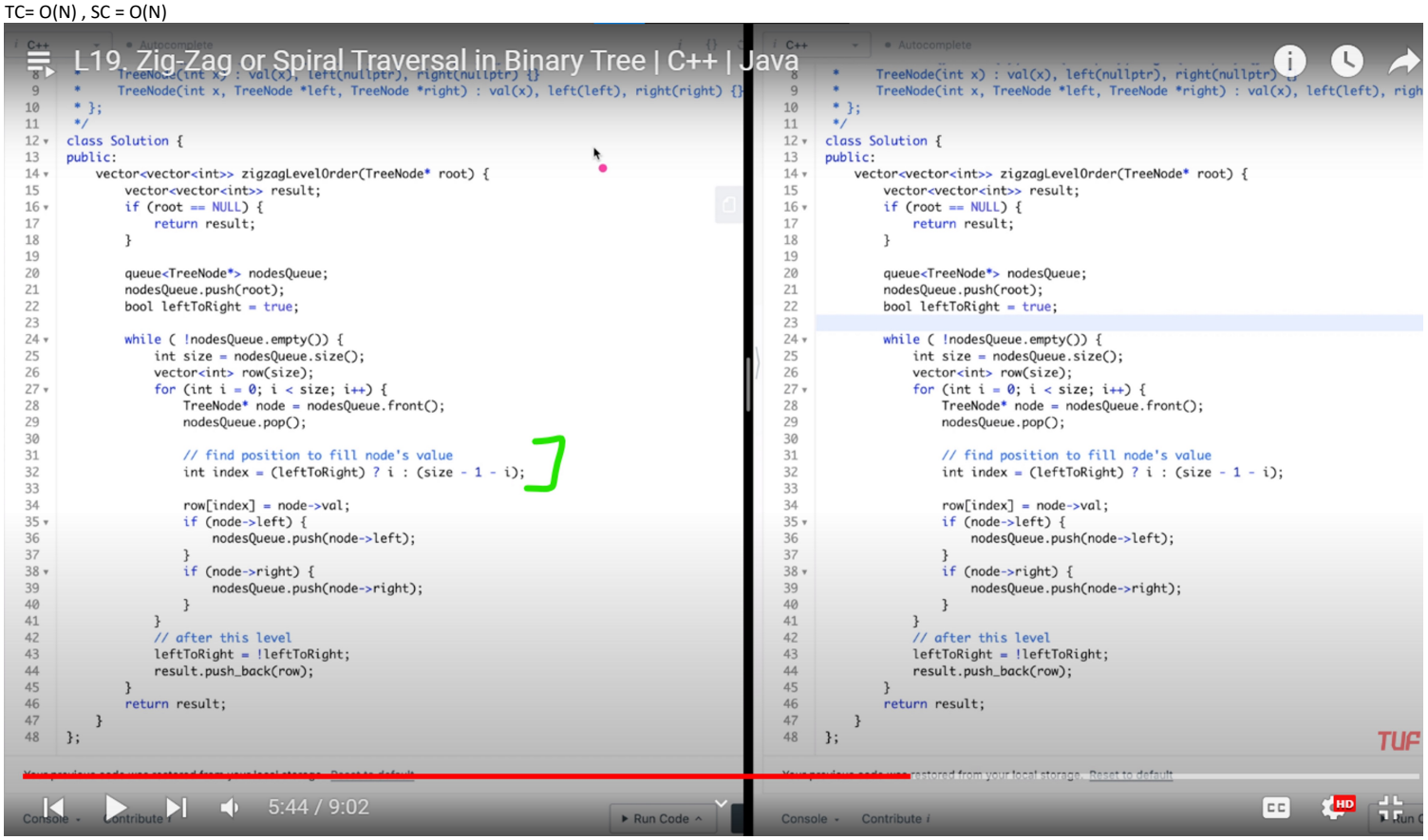
Example 2:

```
/**
 * 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:
    bool isSameTree(TreeNode* p, TreeNode* q) {
        if(p==q && p==nullptr) return true;
        bool ans=true;
        compare(p,q,ans);
        return ans;
    }
    void compare(TreeNode* p, TreeNode* q,bool &ans) {
        if(p==q && p==nullptr) return ;
        if(p==nullptr || q==nullptr || ans==false){ ans=false; return; }
        if(p->val != q->val){ ans=false; return; }
        compare(p->left,q->left,ans);
        compare(p->right,q->right,ans);
    }
};
```

==========

L19.Zig-Zag or Spiral Traversal in binary tree

Striver Solution:
TC= O(N) , SC = O(N)




```
class Solution {
public:
    vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
        vector<vector<int>> result;
        if (root == NULL) {
            return result;
        }

        queue<TreeNode*> nodesQueue;
        nodesQueue.push(root);
        bool leftToRight = true;

        while ( !nodesQueue.empty()) {
            int size = nodesQueue.size();
            vector<int> row(size);
            for (int i = 0; i < size; i++) {
                TreeNode* node = nodesQueue.front();
                nodesQueue.pop();

                // find position to fill node's value
                int index = (leftToRight) ? i : (size - 1 - i);

                row[index] = node->val;
                if (node->left) {
                    nodesQueue.push(node->left);
                }
                if (node->right) {
                    nodesQueue.push(node->right);
                }
            }
            // after this level
            leftToRight = !leftToRight;
            result.push_back(row);
        }
        return result;
    }
};
```

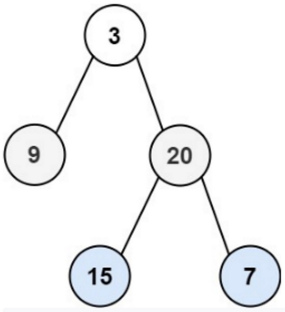
My Solution:
TC,SC : Same as striver's (just that striver did in-place reversal & I ran a loop)

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]
```

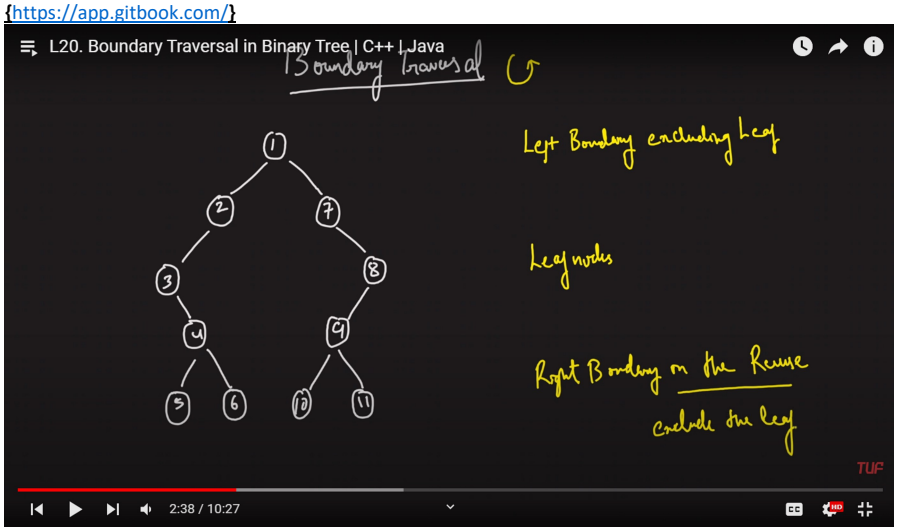
```cpp
//level order traversal ko hi tweek krna h isme
class Solution {
public:
    vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
        vector<vector<int>> ans;
        if(root == nullptr) return ans;

        queue<TreeNode*> q;
        q.push(root);
        ans.push_back({root->val});
        int direction=1;//left to right
        //direction =1 ; //right to left
        while(!q.empty()){
            int n=q.size();
            vector<int> cur;
            for(int i=0;i<n;i++){
                TreeNode* tem=q.front();
                q.pop();
                if(tem->left != nullptr){ q.push(tem->left),cur.push_back(tem->left->val); }
                if(tem->right != nullptr){ q.push(tem->right),cur.push_back(tem->right->val); }
            }
            if(direction==1 ){
                for(int i=0;i<cur.size()/2;i++){
                    int swap=cur[i];
                    cur[i]=cur[cur.size()-i-1];
                    cur[cur.size()-i-1]=swap;
                }
            }
            if(cur.size()>0) ans.push_back(cur);
            direction=1-direction;
        }
        return ans;
    }
};
```

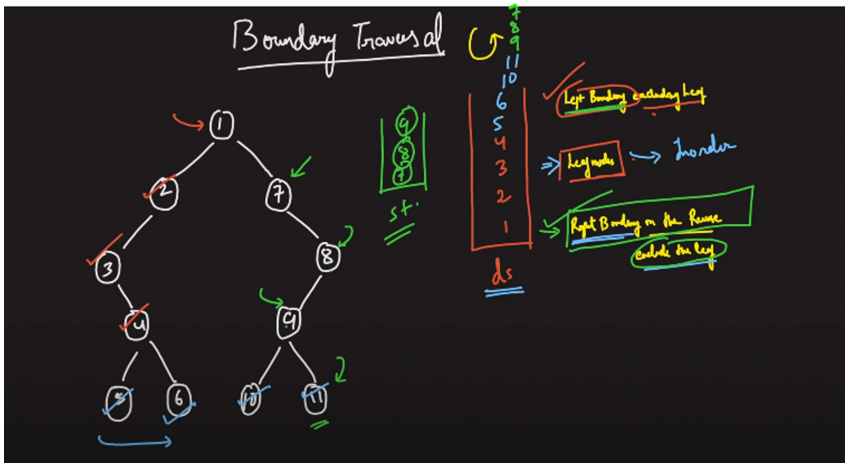
==========

L20. Boundary Traversal in Binary Tree

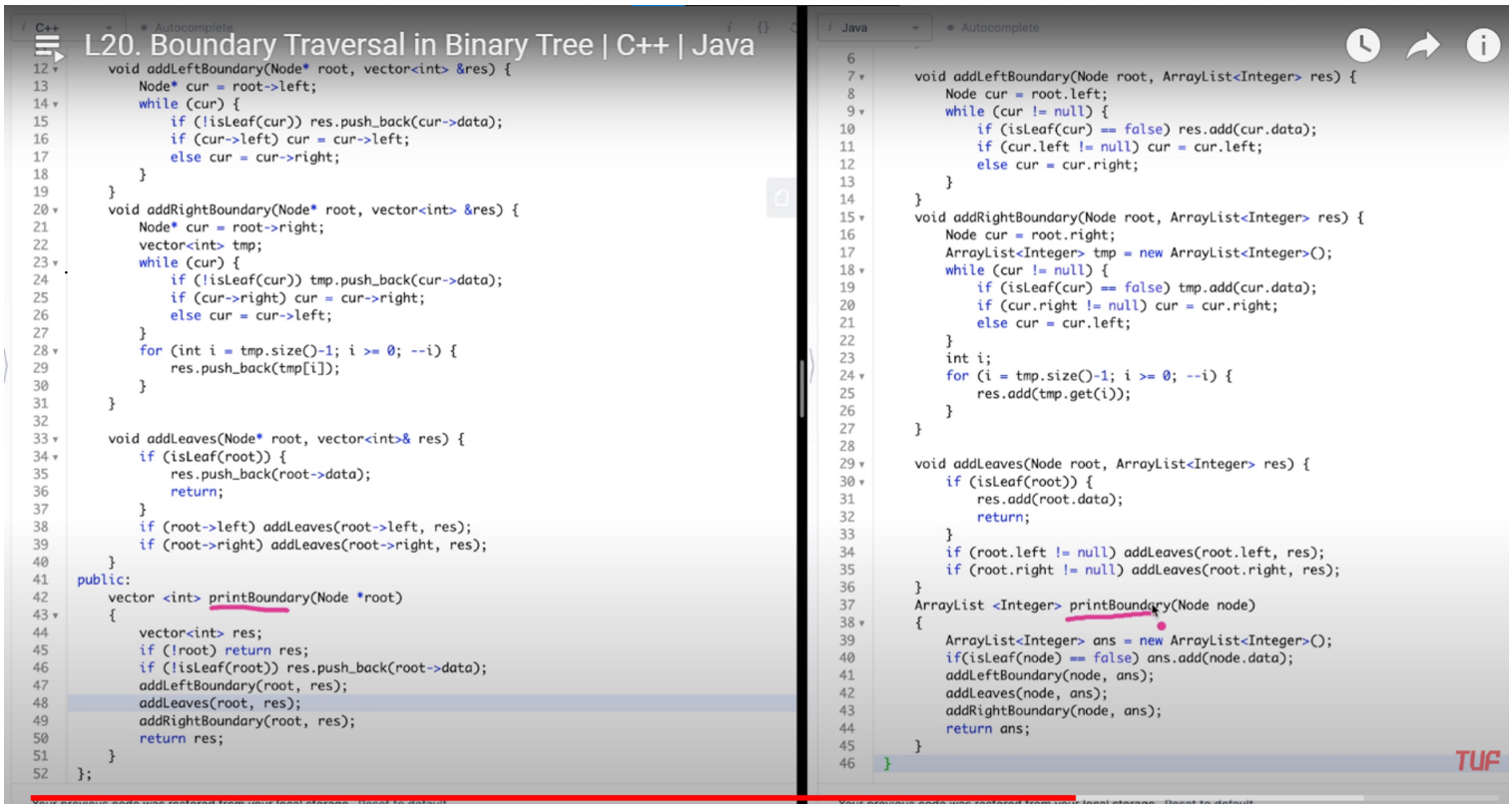
- https://xiaoguan.gitbooks.io/leetcode/content/LeetCode/545-boundary-of-binary-tree-medium.html
- https://leetcode.com/problems/boundary-of-binary-tree/
- https://practice.geeksforgeeks.org/problems/boundary-traversal-of-binary-tree/1#
- https://www.codingninjas.com/codestudio/problems/boundary-traversal_790725

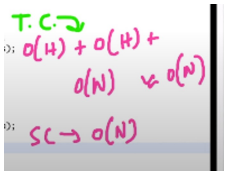
{https://app.gitbook.com/}





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





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 spliting 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
//1 6 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

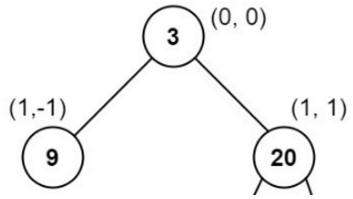
Given the `root` of a binary tree, calculate the **vertical order traversal** of the binary tree.

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

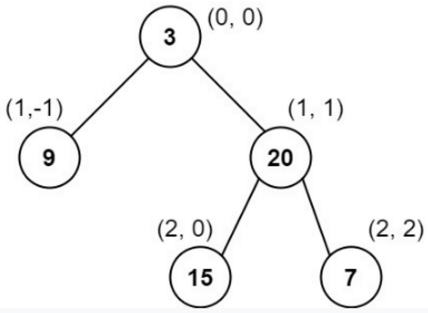
The **vertical order traversal** of a binary tree is a list of top-to-bottom orderings for each column index starting from the leftmost column and ending on the rightmost column. There may be multiple nodes in the same row and same column. In such a case, sort these nodes by their values.

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

Example 1:



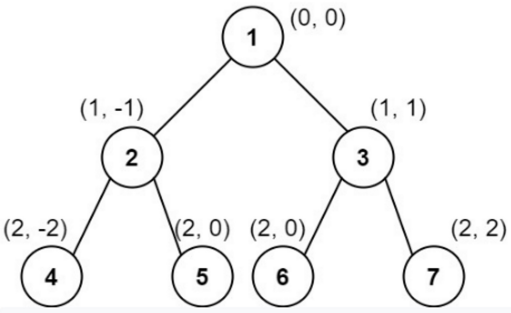
Example 1:



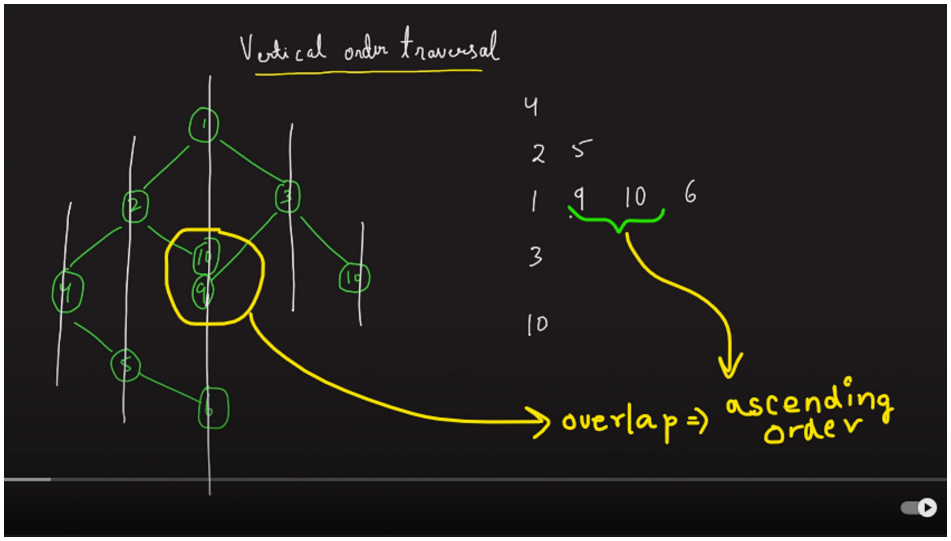

```
Input: root = [3,9,20,null,null,15,7]
Output: [[9],[3,15],[20],[7]]
Explanation:
Column -1: Only node 9 is in this column.
Column 0: Nodes 3 and 15 are in this column in that order from top to bottom.
Column 1: Only node 20 is in this column.
Column 2: Only node 7 is in this column.
```

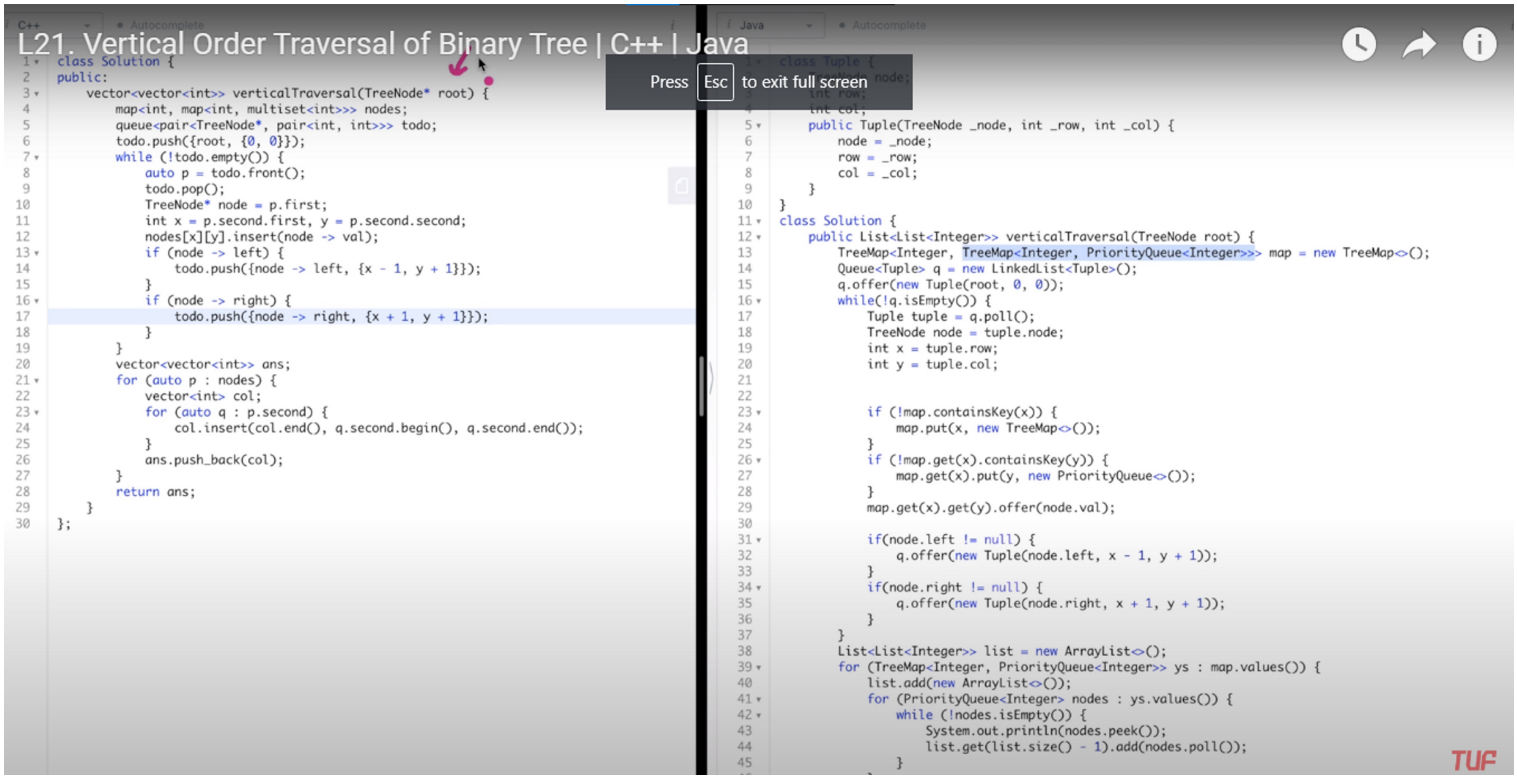
Example 2:

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.
```






```cpp
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;
    }
};
```

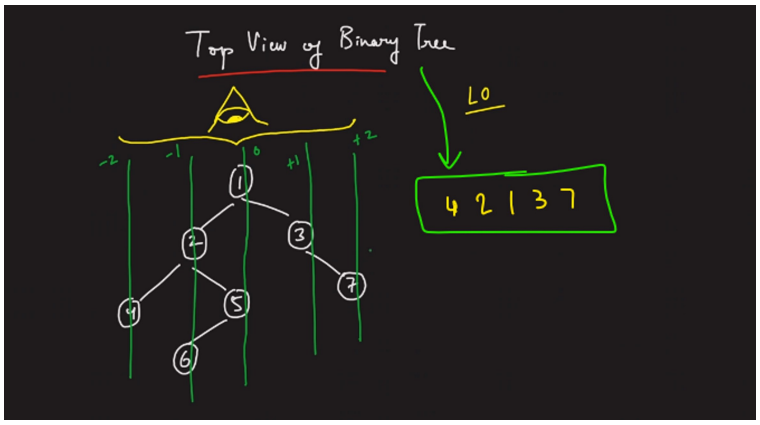
```cpp
class Tuple {
    TreeNode node;
    int row;
    int col;
    public Tuple(TreeNode _node, int _row, int _col) {
        node = _node;
        row = _row;
        col = _col;
    }
}
class Solution {
    public List<List<Integer>> verticalTraversal(TreeNode root) {
        TreeMap<Integer, TreeMap<Integer, PriorityQueue<Integer>>> map = new TreeMap<>();
        Queue<Tuple> q = new LinkedList<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());
                }
```

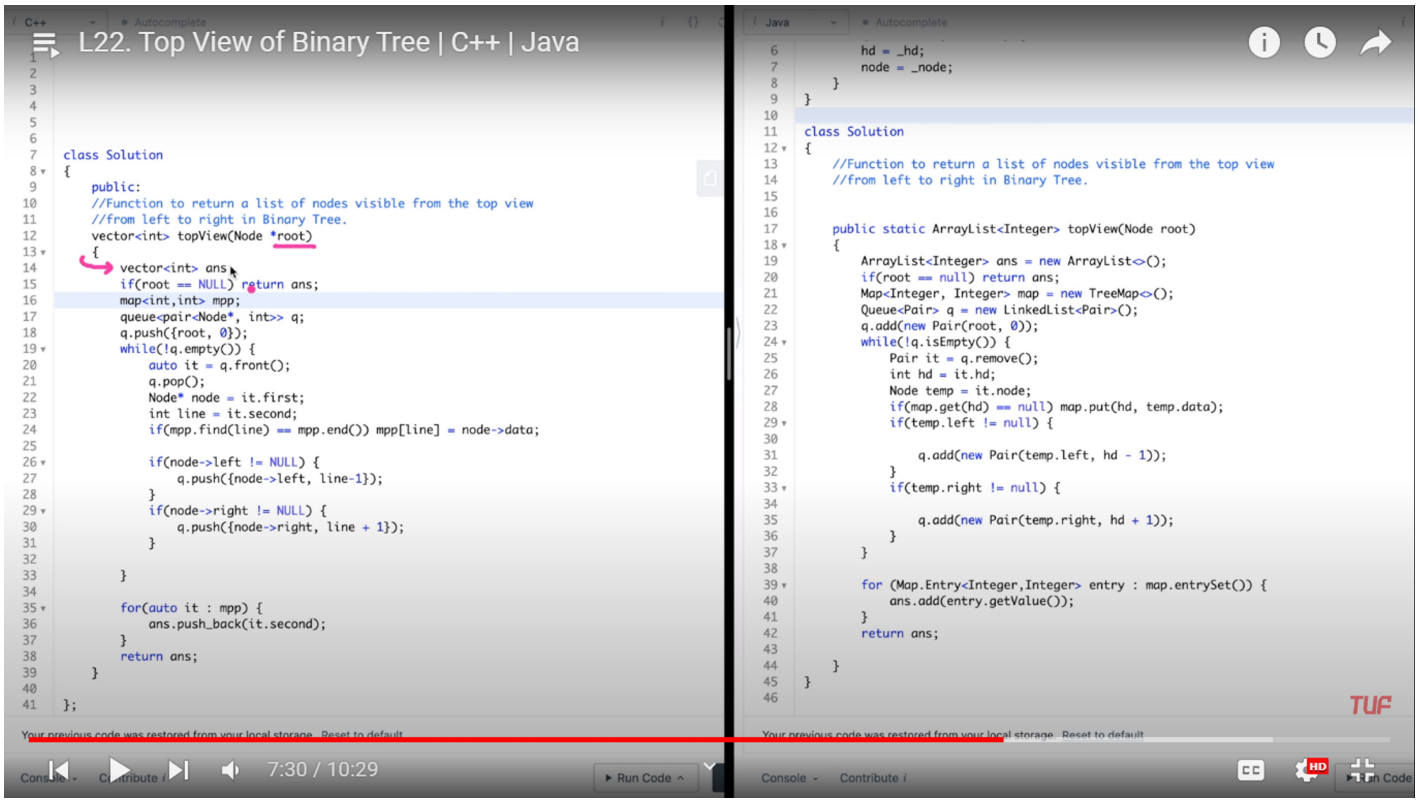
- **Map<int,map<int,multiset<int>>> nodes** => this is :: <x,<y,{set of integers that overlap in sorted order }>>
  - Also note that we use multiset as:
    - 2 or more nodes can have same value
    - we want overlapping nodes in ascending order
- Queue<pair<TreeNode*,pair<int,int>> => this stores :: <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/



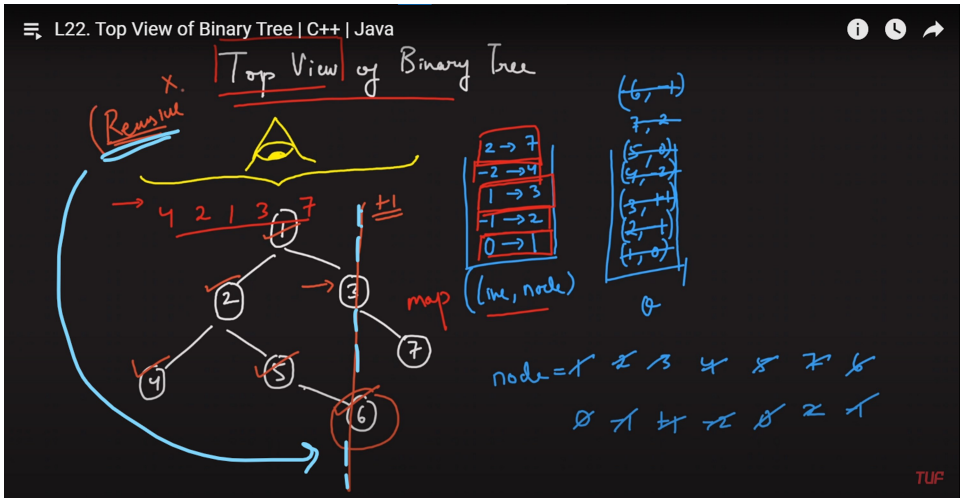


Question : Can we do the top view of binary tree using some recursive traversal ?
Answer : Yes, we can BUT, ***there's a catch,*** we cannot simply use the above algo, but we have to add/consider another parameter of:

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

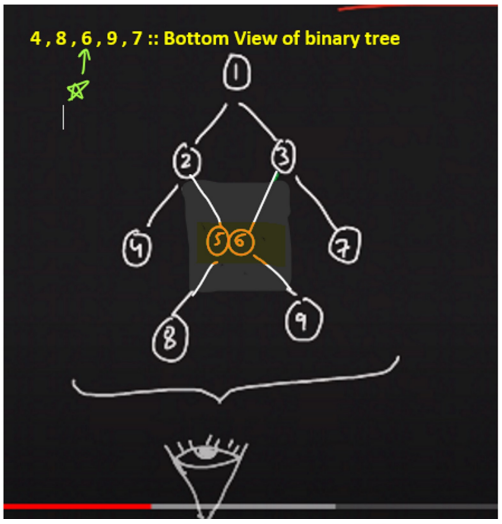
Since in recursive traversal it may happen that the lower node is visited before the upper node and then that ruins the ans ,
so we have to maintain on the same x-co-ordinate , which is the minimum height node and add that node to the top view ans for that x-coordinate

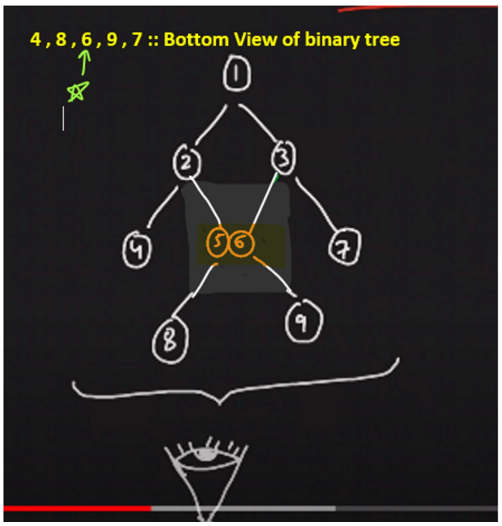
So for this question prefer level-order traversal as its easier to implement as here we don't have to keep track of the depth of the node
But if we implement using recursive traversal then we have to implement the extra logic of depth of node.



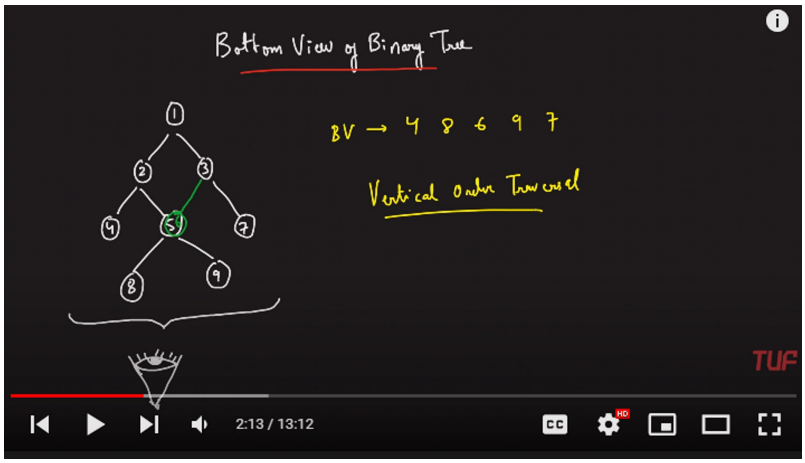
==========

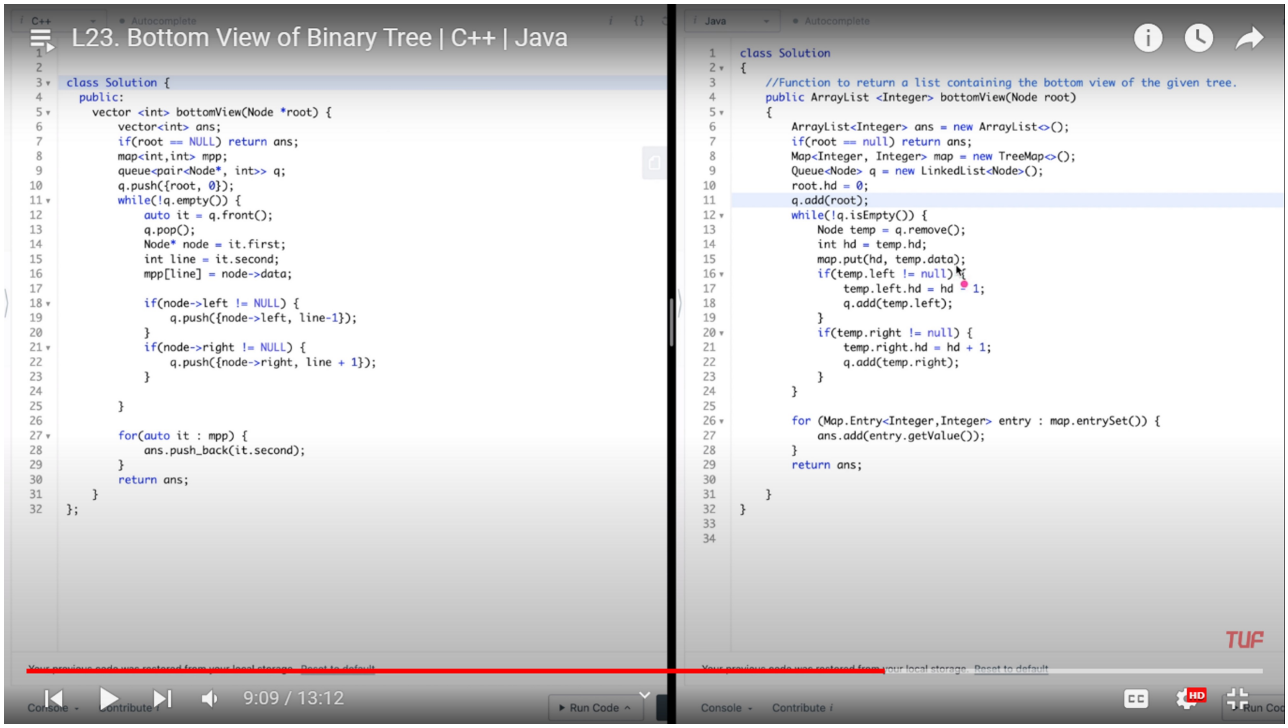
L23. Bottom view of binary tree





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!





My Code : with pblm 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
```

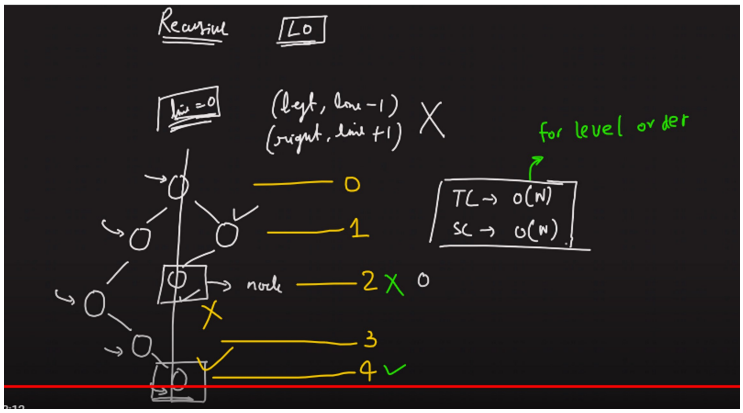
For the above tree, the bottom view is 5 10 3 14 25.
If there are **multiple** bottom-most nodes for a horizontal distance from root, then print the later one in level traversal. For example, in the below diagram, 3 and 4 are both the bottommost nodes at horizontal distance 0, we need to print 4.

```
        20
```

```
94  //Function to return a list containing the bottom view of the given tree.
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         }return ans;
117     }
118 };
119
```

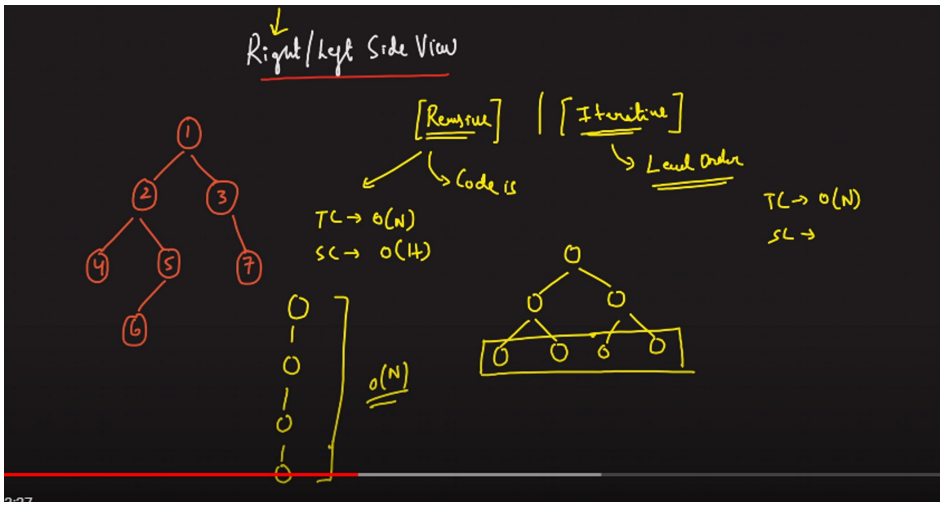
Question: Will recursive traversal work for this question?
Answer : Yes it will work BUT not directly.
We will have to consider the depth of the current node wrt the root and the highest depth node will be the answer for a x-co-ordinate!
So for this question prefer level-order traversal as its easier 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

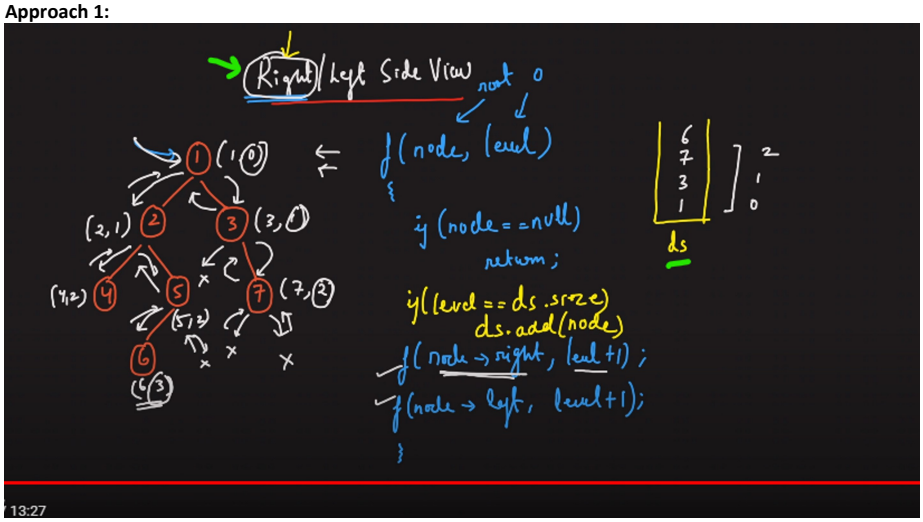


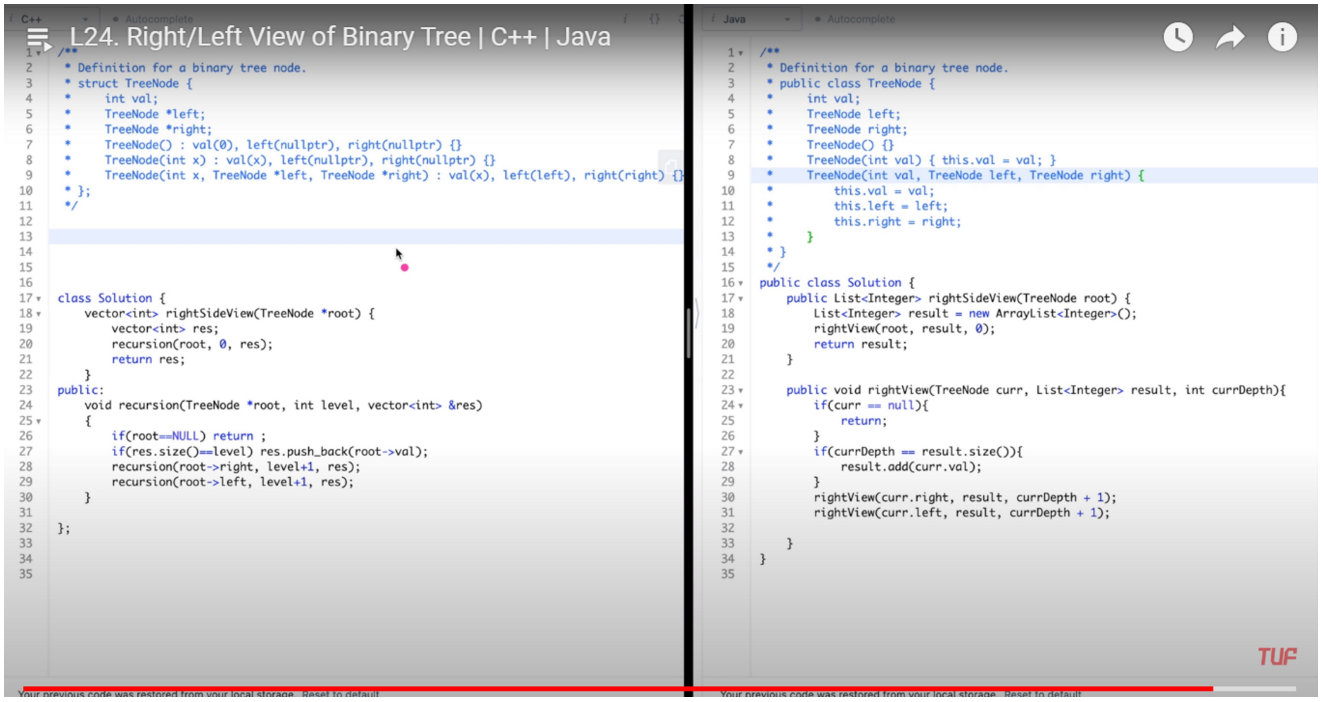
2 Approaches:

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(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:





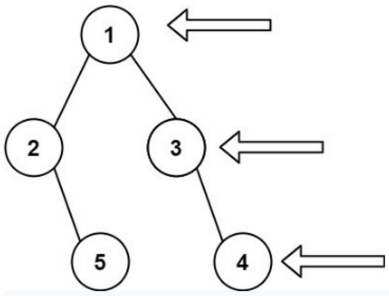
Approach 2:
My Approach : Level Order Traversal
TC : O(N) , Auxillary SC = O(N)

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]
```

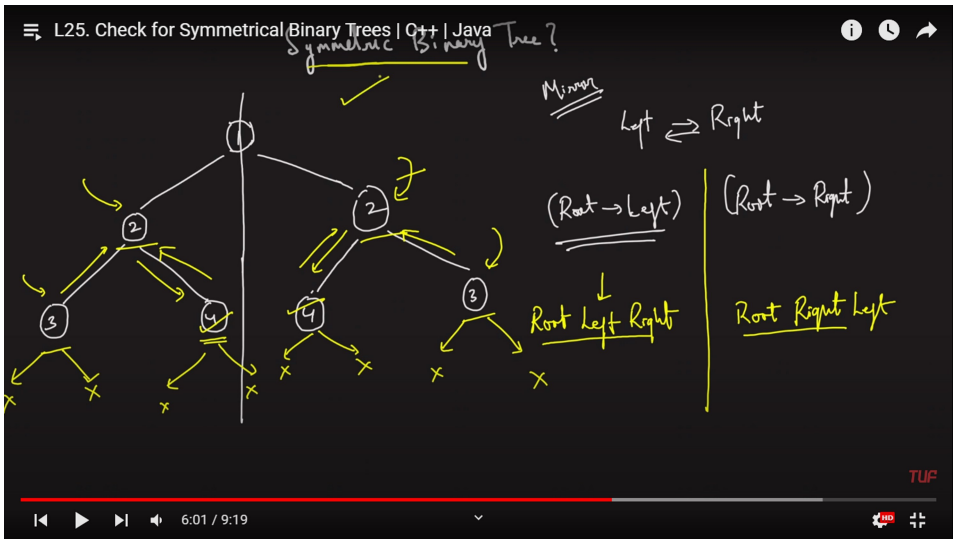
Example 2:

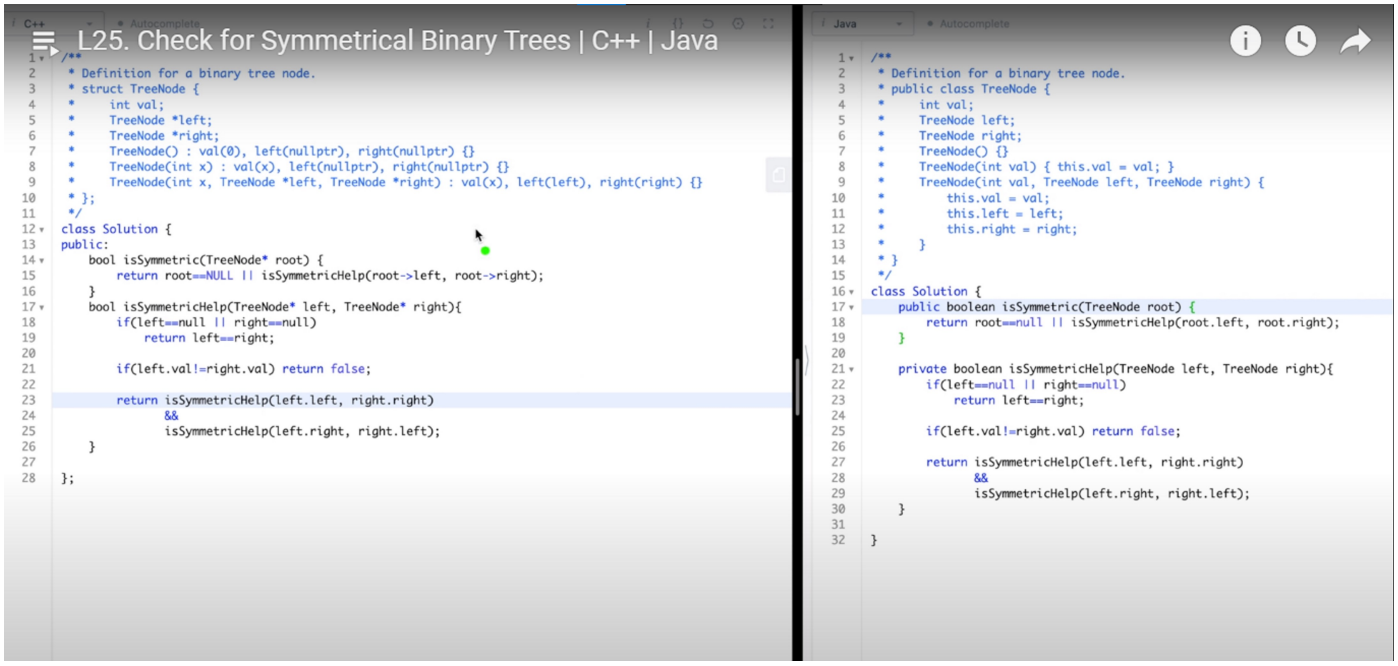
```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    vector<int> rightSideView(TreeNode* root) {
        //at each level , the x-max
        //level-order traversal =>last element to be entered
        vector<int> ans;
        if(root==nullptr) return ans;
        queue<TreeNode*> q;
        q.push(root);
        while(!q.empty()){
            int n = q.size();
            for(int i=0;i<n;i++){
                if(i==n-1) ans.push_back(q.front()->val);//for right side view
                // if(i==0) ans.push_back(q.front()->val);//for left side view
                TreeNode* tem=q.front();
                q.pop();
                if(tem->left != nullptr) q.push(tem->left);
                if(tem->right != nullptr) q.push(tem->right);
            }
        }
        return ans;

    }
};
```

==========

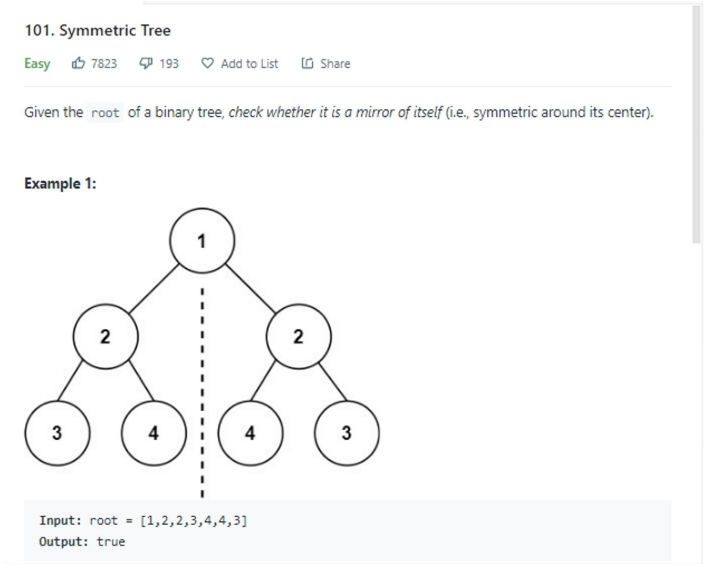
L25. Symmetric Tree

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





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




```
/**
 * 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:
//     TC=O(n) , SC=O(n)
    bool isSymmetric(TreeNode* root) {
        bool ans=true;
        mirror(root->left,root->right,ans);
        return ans;
    }
    void mirror(TreeNode* left,TreeNode* right,bool& ans){
        if(left==nullptr && right==nullptr) return;
        if(left==nullptr || right==nullptr){ ans=false;return; }
        if(left->val!=right->val){ ans=false;return; }

        mirror(left->left,right->right,ans);
        mirror(left->right,right->left,ans);
    }
};
```

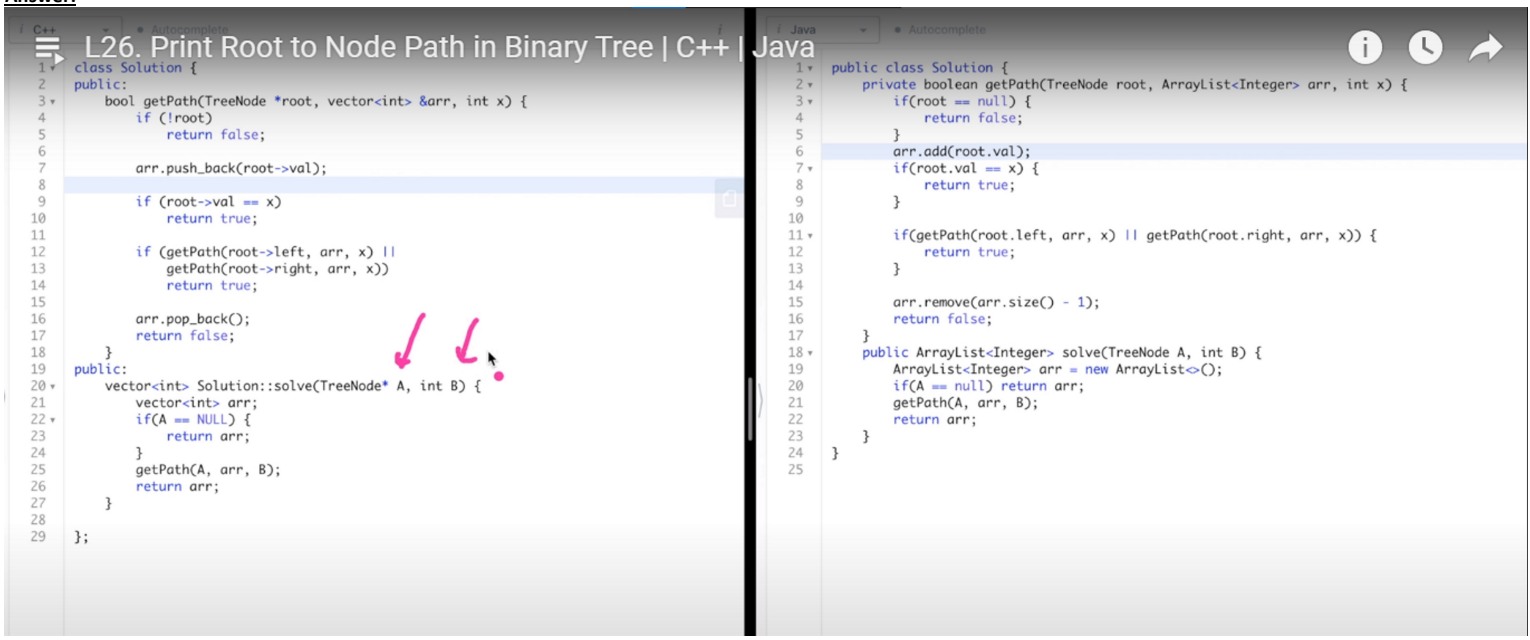
Your previous code was restored from your local storage. Reset to default

==========

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

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

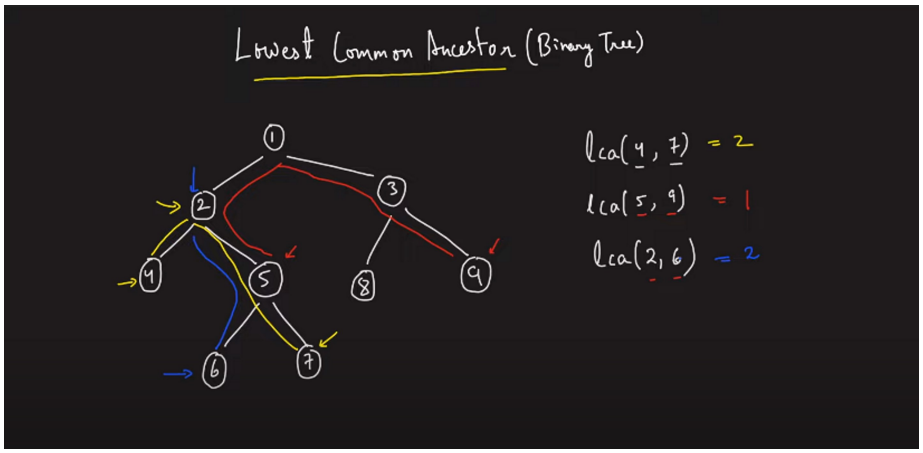
Answer:

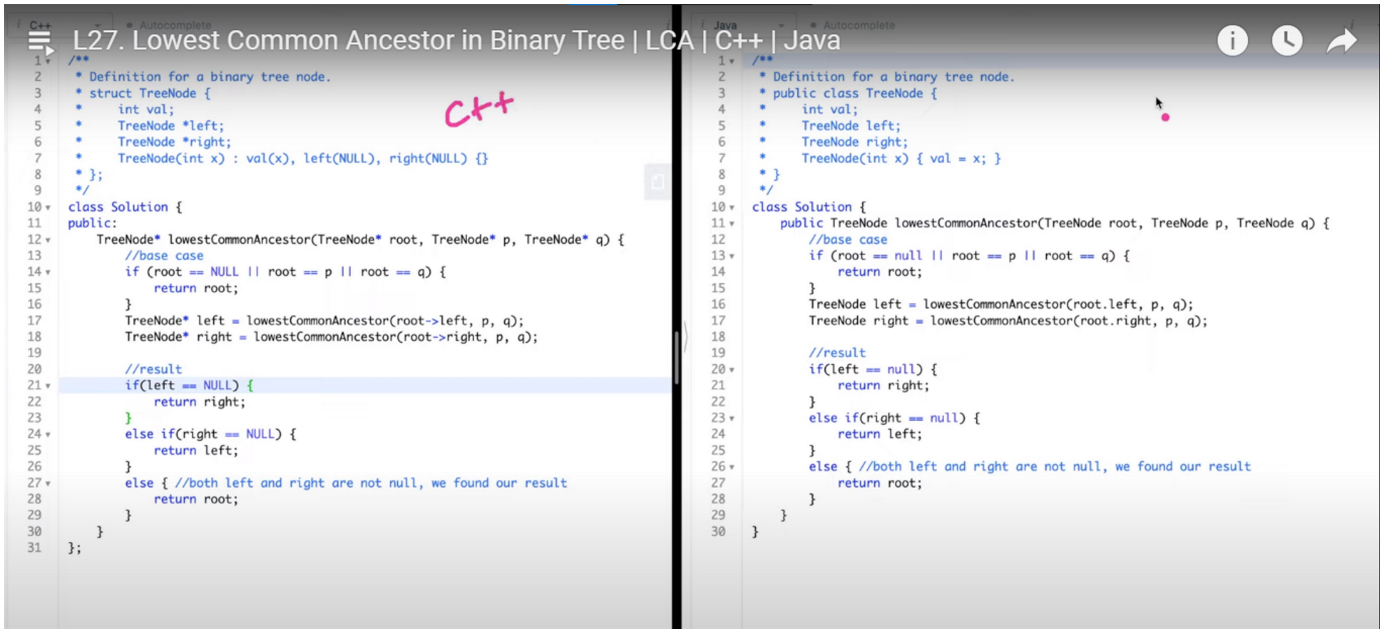



```
bool getPath(TreeNode *root, vector<int> &arr, int x) {
   // if root is NULL
   // there is no path
   if (!root)
      return false;

   // 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.






```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
        //base case
        if (root == NULL || root == p || root == q) {
            return root;
        }
        TreeNode* left = lowestCommonAncestor(root->left, p, q);
        TreeNode* right = lowestCommonAncestor(root->right, p, q);

        //result
        if(left == NULL) {
            return right;
        }
        else if(right == NULL) {
            return left;
        }
        else { //both left and right are not null, we found our result
            return root;
        }
    }
};
```

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

        //result
        if(left == null) {
            return right;
        }
        else if(right == null) {
            return left;
        }
        else { //both left and right are not null, we found our result
            return root;
        }
    }
}
```

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

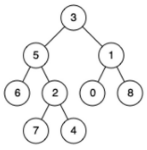
236. Lowest Common Ancestor of a Binary Tree

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.
```

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
        //in-order traversal
        //base case
        if(root==nullptr || root==p || root==q) return root;

        //left/right stores:
        // -either null (if neither p nor q was found along the left subtree)
        // -or, one of the nodes to be found :p / q
        TreeNode* left=lowestCommonAncestor(root->left,p,q);
        TreeNode* right=lowestCommonAncestor(root->right,p,q);

        //while backtracking (result)
        if(left==nullptr) return right;
        else if(right==nullptr) return left;
        else return root;//if neither left nor right is null => root is the LCA


    }
};
```

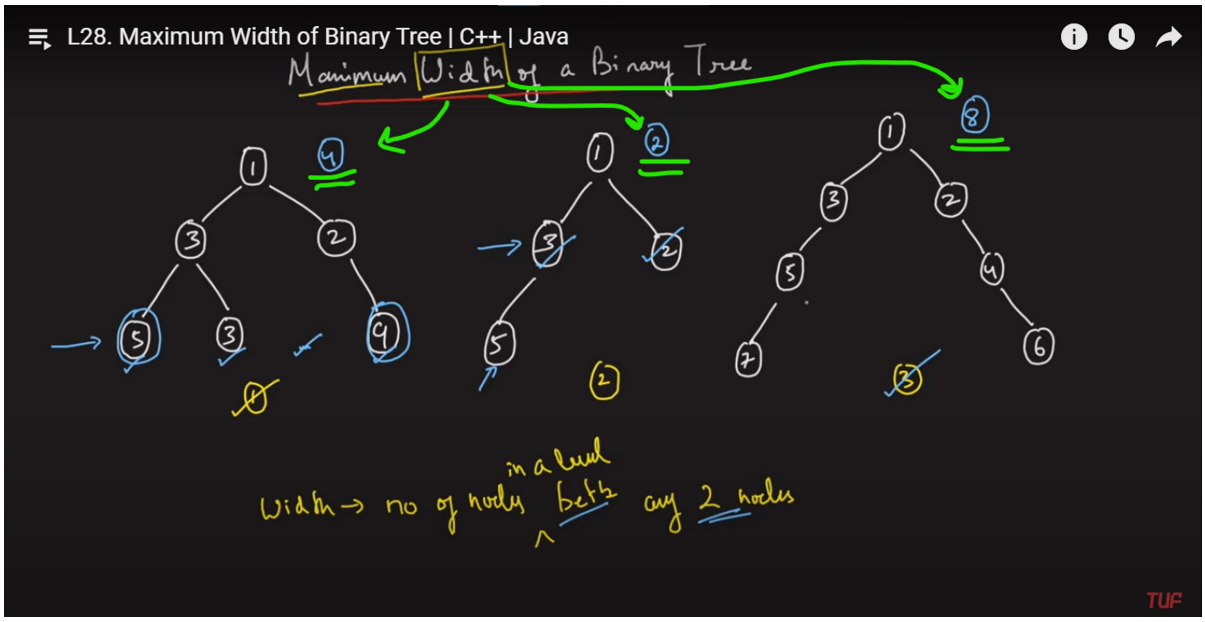
Note :

- we can prune the backtrack if on a node we get both left and right as NOT NULL, and prune it when this type if ode 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/

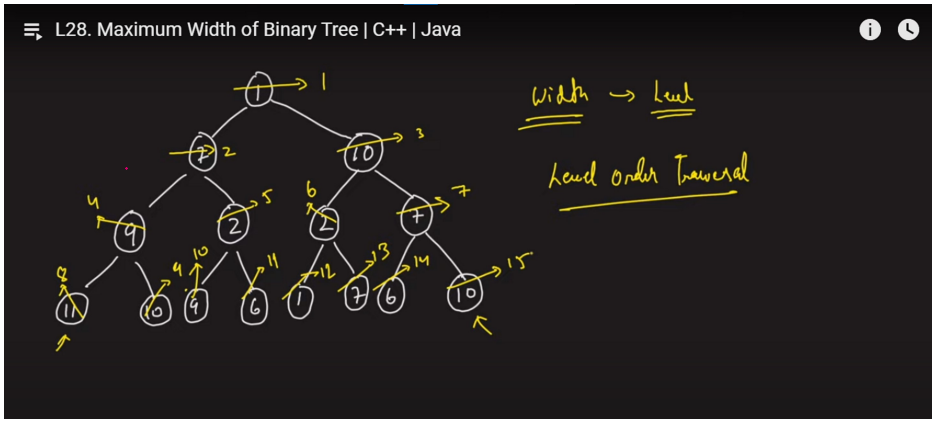


Width of binary tree: number of nodes in a level between the 2 extreme nodes present(including those nodes/extreme nodes).

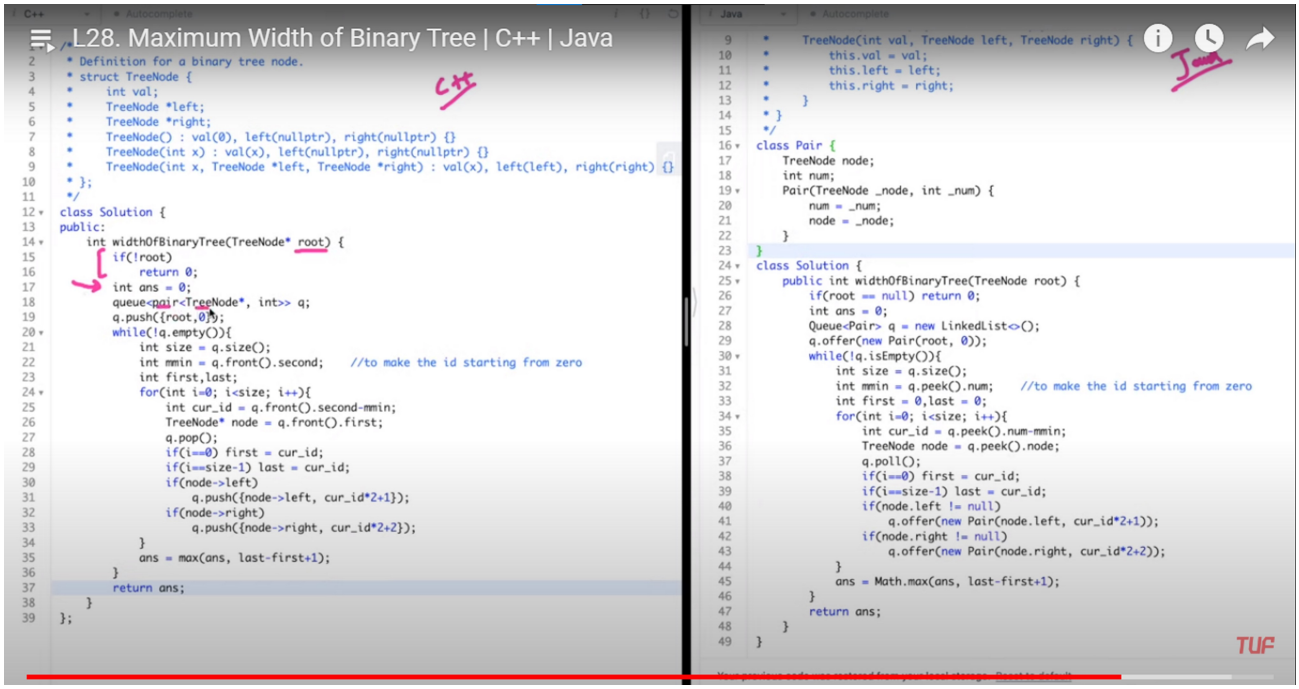
- Note the extreme nodes must be present in the binary tree to count the nodes in between them(between nodes may/may not exist)

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:

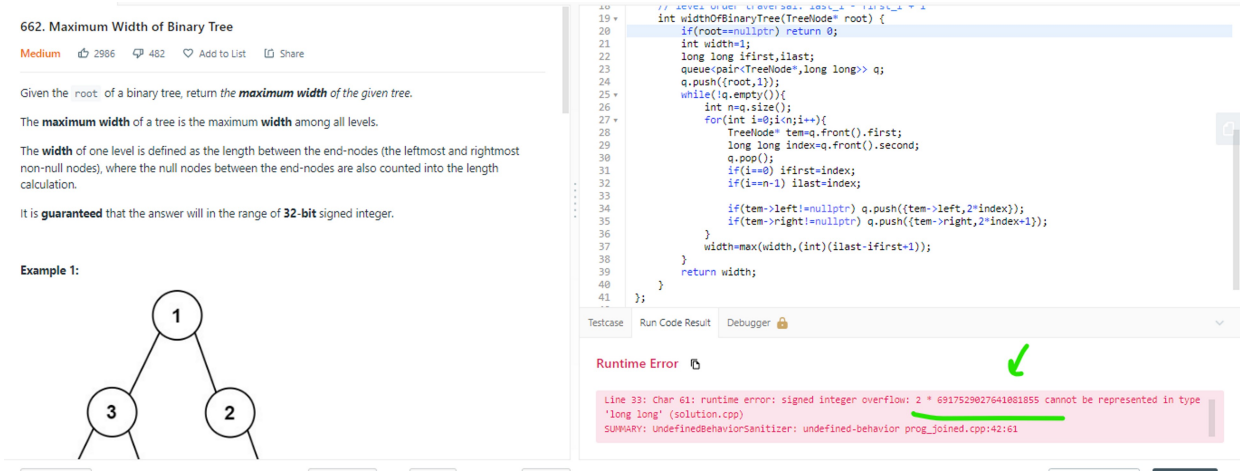


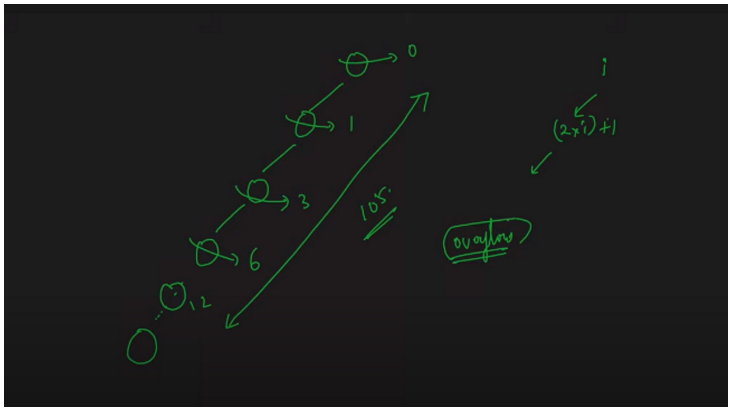
My initial Solution:

My Code is easier to understand for me at least!

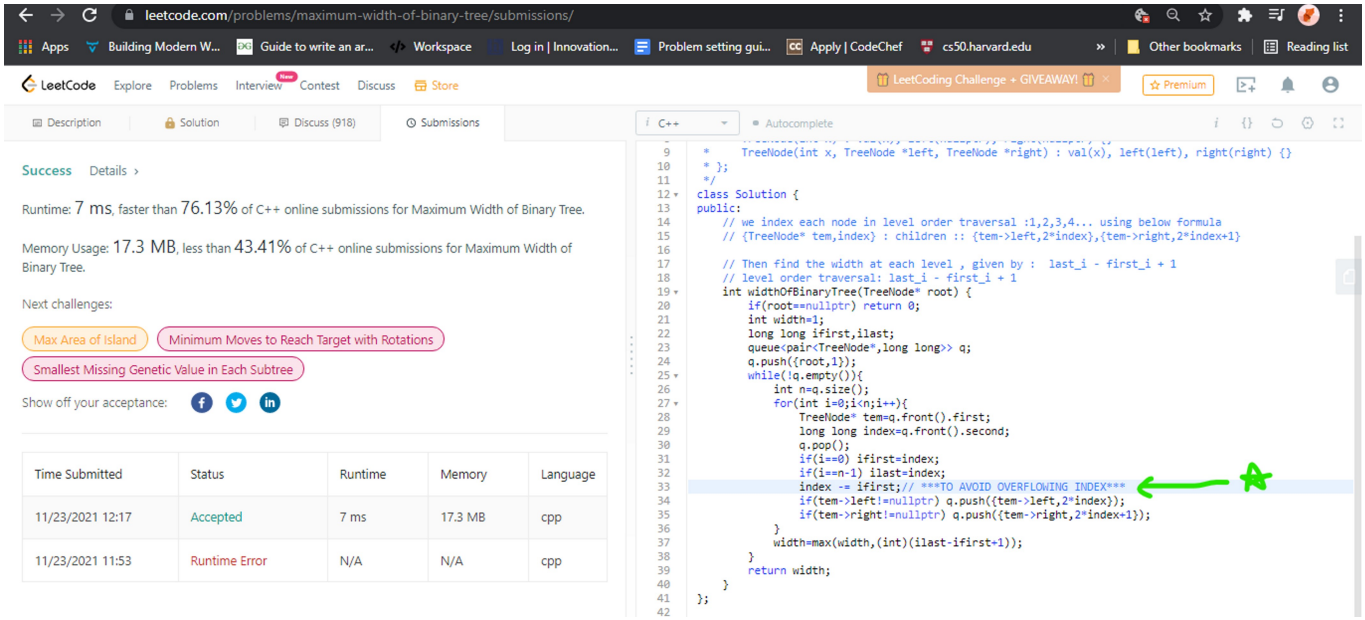
Note: the approach is correct here , but there will be an overflow error for indexing if the tree is skewed (max: 2^3000, can be the index)

- so we have to find a way to index the nodes to avoid overflow
- We do that by
  - Index : {index - leftmost_index_on_that_level}





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



==========

L29.

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

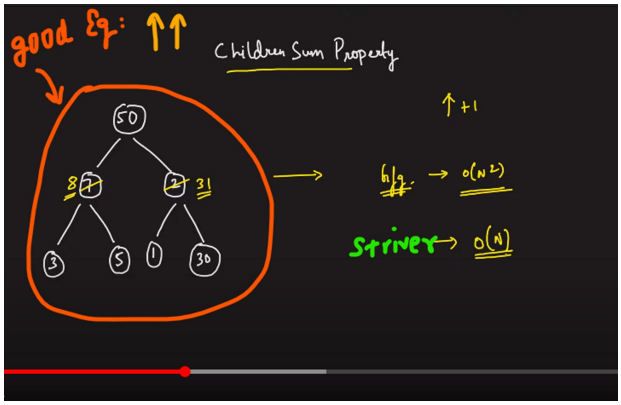
Question Link: https://www.codingninjas.com/codestudio/problems/childrensumproperty_790723?source=youtube&campaign=Striver_Tree_Videos&utm_source=youtube&utm_medium=affiliate&utm_campaign=Striver_Tree_Videos

NOTE/WARNING:
This question is NOT as EASY as it seems

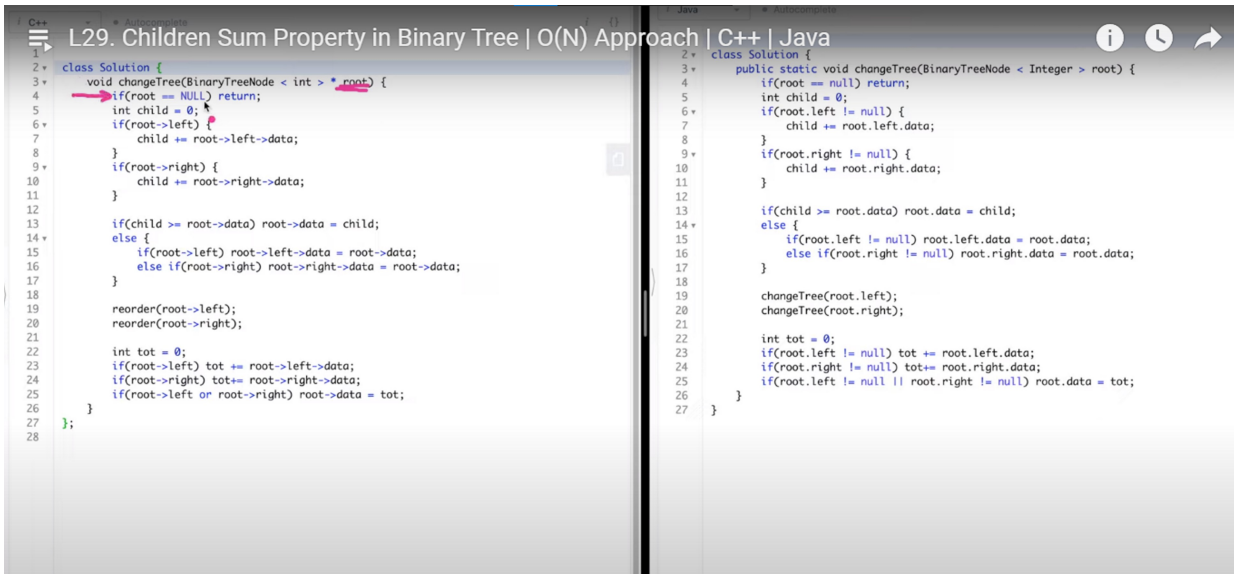
GFG solution: O(N*N) :: https://www.geeksforgeeks.org/convert-an-arbitrary-binary-tree-to-a-tree-that-holds-children-sum-property/?ref=rp
Striver's Solution: O(N) :: https://github.com/striver79/FreeKaTreeSeries/blob/main/L29_childrenSumProblemCpp

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:




```
void reorder(BinaryTreeNode < int > * root) {
    if(root == NULL) return;
    int child = 0;
    if(root->left) {
        child += root->left->data;
    }
    if(root->right) {
        child += root->right->data;
    }

    if(child >= root->data) root->data = child;
    else {
        if(root->left) root->left->data = root->data;
        else if(root->right) root->right->data = root->data;
    }

    reorder(root->left);
    reorder(root->right);

    int tot = 0;
    if(root->left) tot += root->left->data;
    if(root->right) tot+= root->right->data;
    if(root->left or root->right) root->data = tot;
}
void changeTree(BinaryTreeNode < int > * root) {
    reorder(root);
}
```

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

codestudio Practice Guided Path Interview Prep Challenges New Knowledge Centre New Community New

Problem Submissions Solution New Discuss

Children Sum Property

8 Difficulty: MEDIUM

Contributed By Ayush Thakur

Avg. time to solve 36 min

Success Rate 70%

Problem Statement Suggest Edit

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.

Previous Next

C++ (g++ 5.4)

```
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 }
```

Console

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

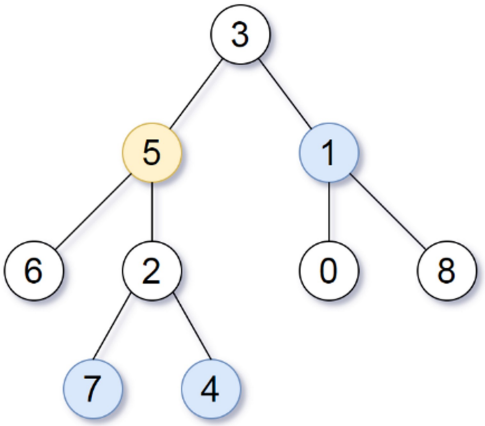
Medium 5057 99 Add to List Share

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 <= Node.val <= 500`
- All the values `Node.val` are **unique**.
- `target` is the value of one of the nodes in the tree.
- `0 <= k <= 1000`

Accepted 186,562 | Submissions 311,208

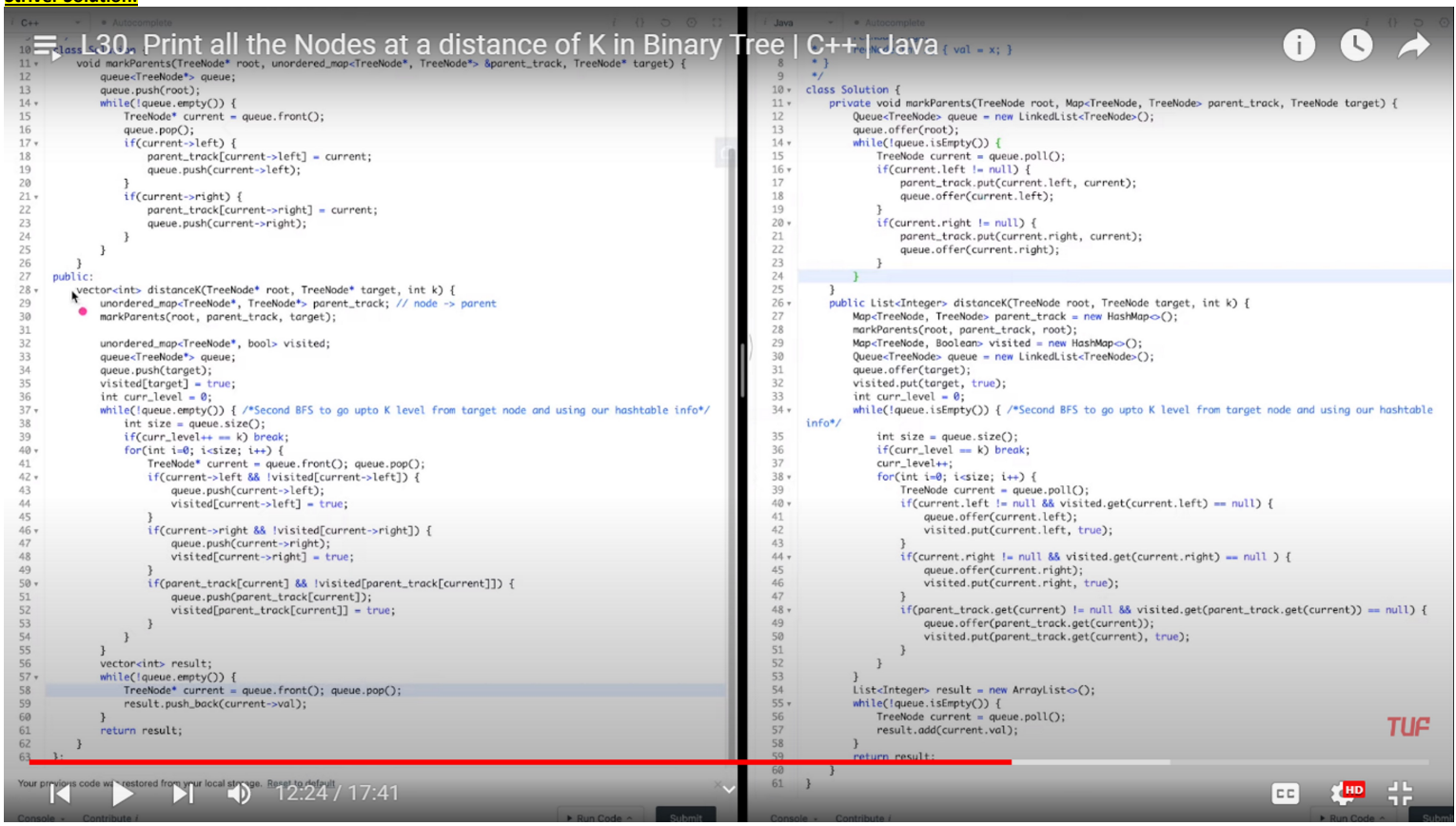
Seen this question in a real interview before? Yes No

Companies

Related Topics

Tree | Depth-First Search | Breadth-First Search | Binary Tree

Striver Solution:



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=dpeth 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.