

Today's Checklist

- ✓ What is a Tree data structure
- ✓ Representation
- ✓ Terminology
- Important properties of trees
- Types of Trees
- Applications of tree data structure
- What is a Binary Tree?
- Implementation
- Traversals
- Problems

Types of Binary Trees

What is a Tree data structure

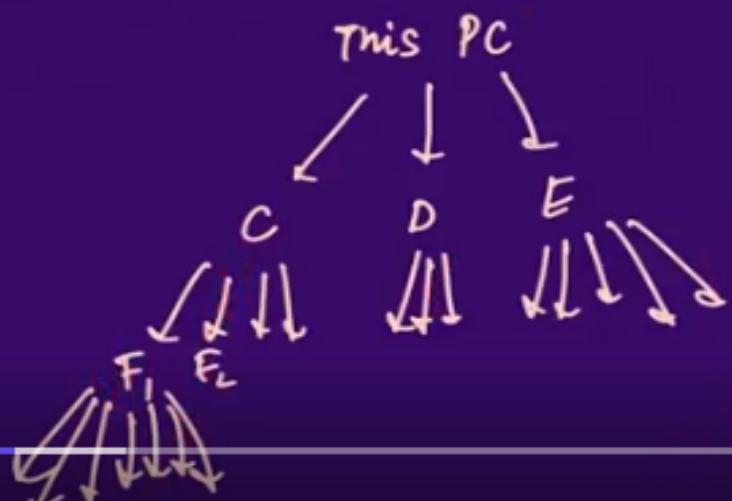
Array, LL, Stack, Queue → Linear data structures

What is a Tree data structure

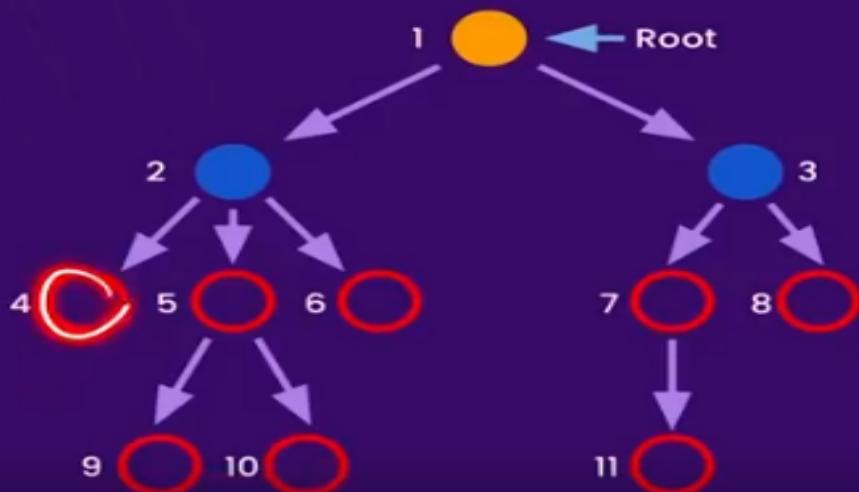
Array, LL, Stack, Queue → Linear data structures

Tree → hierarchical data structure

Co.



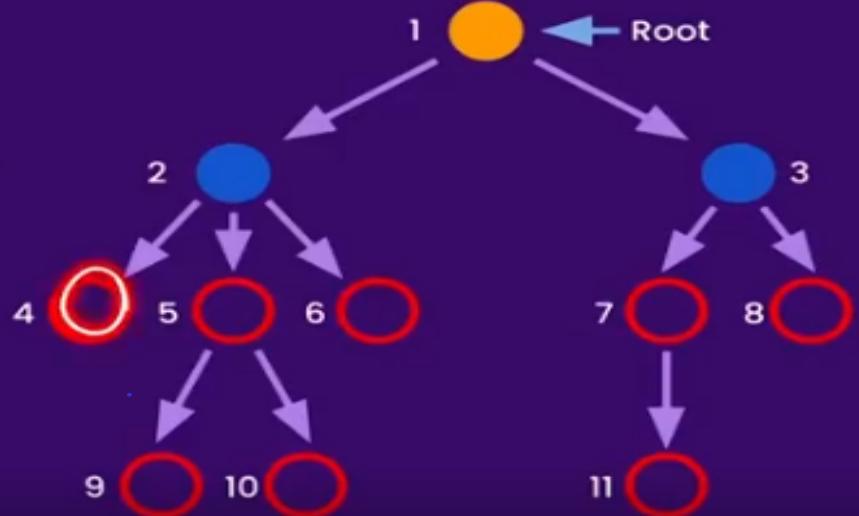
Representation



Terminology

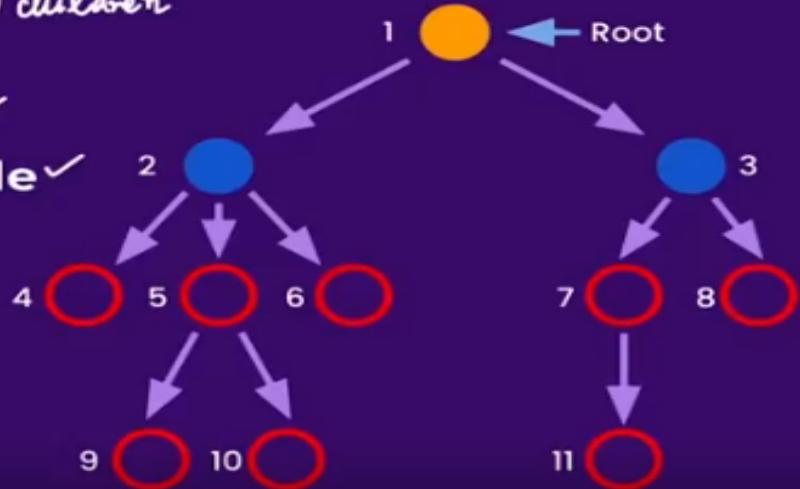
- **Root** ✓
- **Child Node** ✓
- **Parent Node** ✓
- **Sibling Nodes** ✓

means nodes
with a common
parent



Terminology

- **Leaf Node** → with 0 children
- **Internal Node**
- **Ancestor Node** ✓
- **Descendant Node** ✓



1 be ancestor → 1, 2, 5

2 be descendants : 4, 5, 6, 9, 10

Internal node-->ye na to saroundig or na to leaf node hota hai .

Leaf node-->no child.

Root node-->ye vooh node hai jiska koi parent node nhi hota hai iske keval child node hote hai.

Ancestor node --> this node like purvaj.

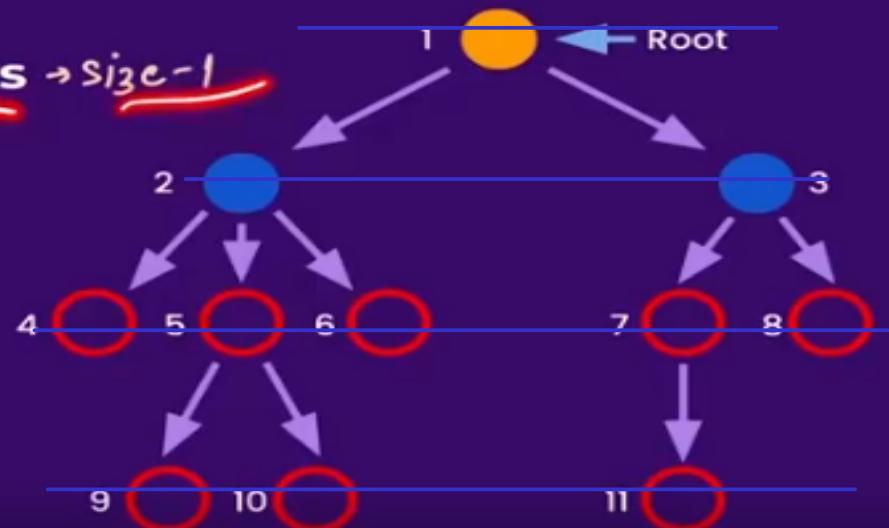
Terminology

• **Level**

• **Number of edges** \rightarrow size - 1

• **Height** : levels - 1

• **Size** : no. of nodes



$$\text{levels} = 4$$

$$\text{Size} = 11$$

$$\text{Edges} = 10$$



Leval-->level like a generation.

Number of edges=size-1.

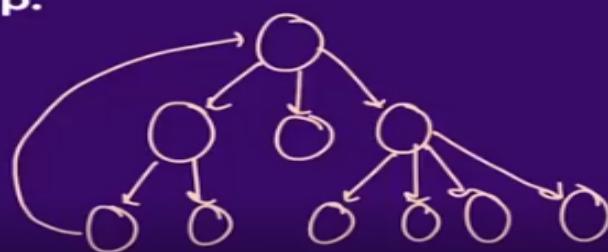
height = levels-1.

size=number of nodes in a tree

Important Properties of trees



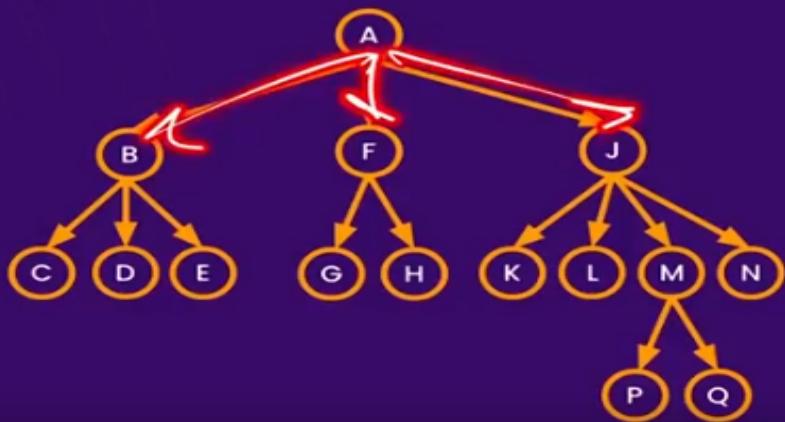
- Traversing in a tree is done by depth first search and breadth first search algorithm.
- It has **no loop** and **no circuit**.
- It has **no self-loop**.



Not a Tree



1. Generic Trees

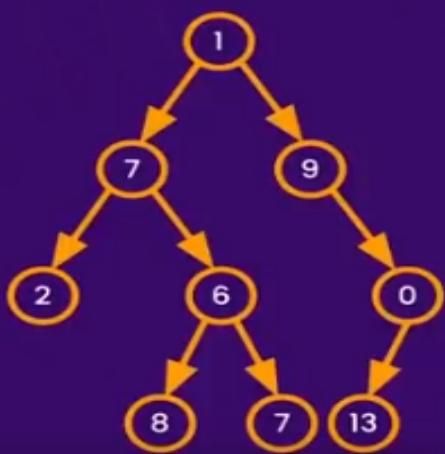


Any node in a generic Tree can have any no. of child nodes



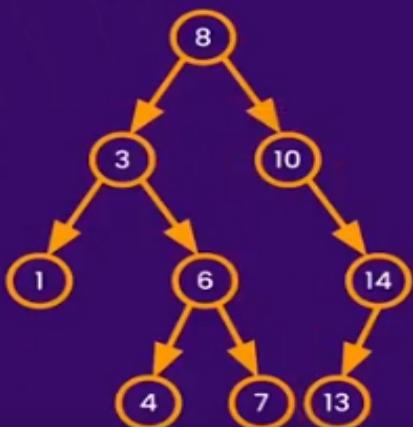
2. Binary Trees

Any node can have 0, 1 or 2 children



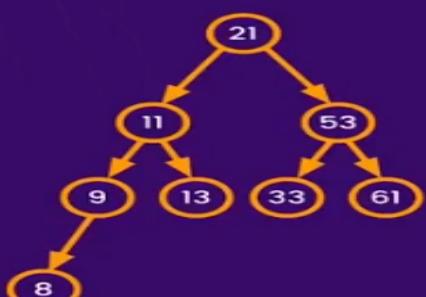
3. Binary Search Trees

ISke baare me baad me acche se badhege

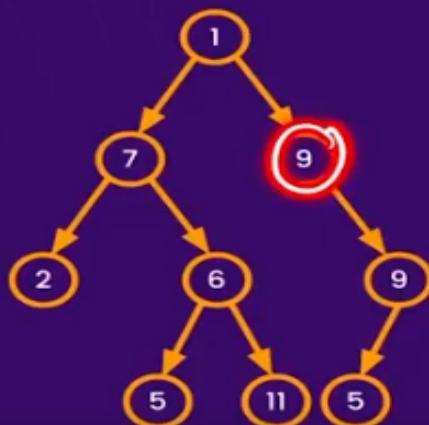


4. AVL Trees

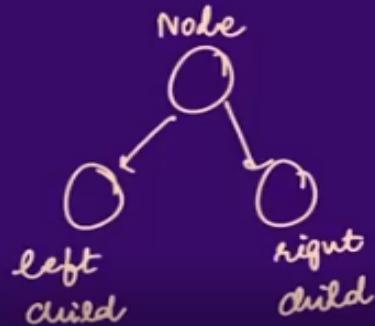
BST's which are self balancing



What are Binary Trees?



Trees where each node has either 0, 1 or 2 children



Implementation

- Creating a Node class

```
class Node {  
    int val;  
    Node left;  
    Node right;
```

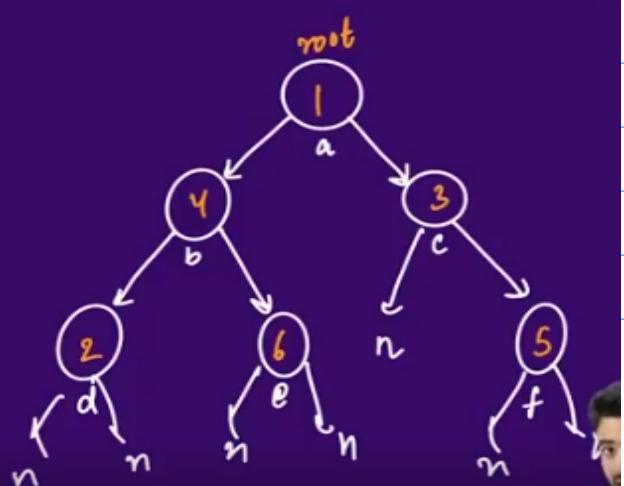
Creating a Node class

```
class Node {  
    int val;  
    Node left;  
    Node right;
```

Implementation

- Creating a Node class

```
class Node {  
    int val;  
    Node left;  
    Node right;
```



```

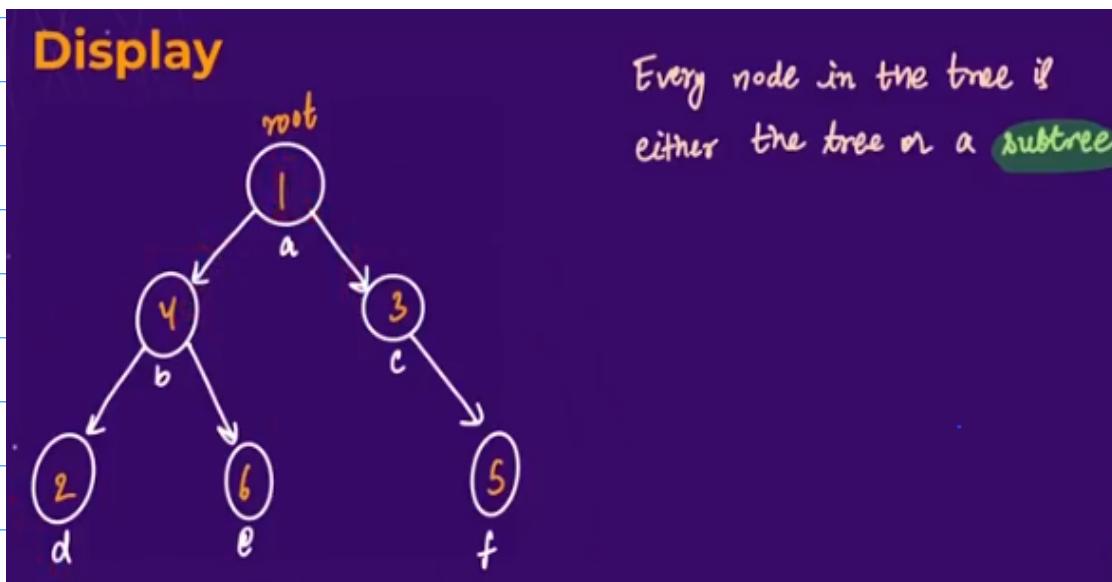
public static void main(String[] args) {
    Node a = new Node(1);
    Node b = new Node(4);
    Node c = new Node(3);
    Node d = new Node(2);
    Node e = new Node(6);
    Node f = new Node(5);
    a.left = b;a.right = c;
    b.left = d;b.right = e;
    c.right =f;
    System.out.println(e.val);
    System.out.println(a.left.right.val);
    System.out.println(a.right.right.val);
}

```

subtree means part of the tree

To Display

- 1) Ensure a base case
- 2) Print root's val
- 3) Recursion will print LST & RST



```

static private void display(Node root){
    if(root==null) return;
    System.out.print(root.val+" ");
    display(root.left);
    display(root.right);
}

public static void main(String[] args) {
    Node a = new Node(1);
    Node b = new Node(4);
    Node c = new Node(3);
    Node d = new Node(2);
    Node e = new Node(6);
    Node f = new Node(5);
    a.left = b;a.right = c;
    b.left = d;b.right = e;
    c.right =f;
    a.left.right.val = 56;
    display(a);
}

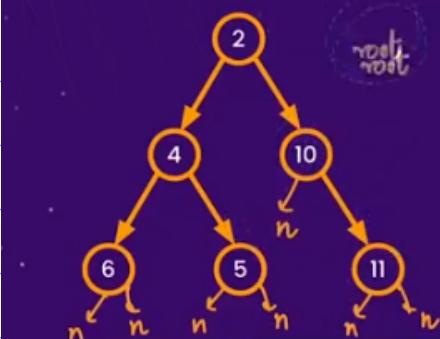
```

-->**output**

1 4 2 56 3 5

Process finished with exit code 0

Display



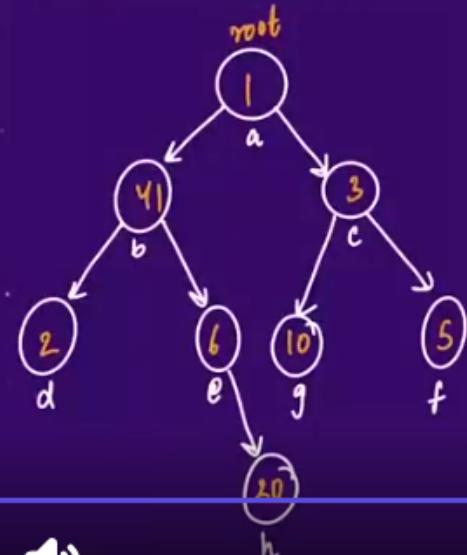
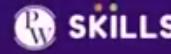
Output

2 4 6 5 10 11

```
private static void display(Node root){  
    if(root==null) return; // Base Case  
    System.out.print(root.val+" ");  
    display(root.left); // Left SubTree  
    display(root.right); // Right SubTree  
}
```



Find sum of tree nodes



```
if(node==null) return 0;  
Sum = root.val + sum(LST) + sum(RST)
```



Homework :

1. Find product of tree nodes
2. Find product of non-zero elements of nodes

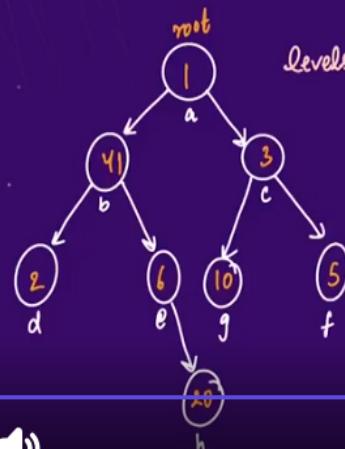
package Tree;

```
public class findProductOfTreeNode {  
    static private int product(Node root){  
        if(root==null) return 1;  
        return root.val*product(root.left)*product(root.right);  
    }
```

```
public static void main(String[] args) {  
    Node a = new Node(1);  
    Node b = new Node(4);  
    Node c = new Node(3);  
    Node d = new Node(2);  
    Node e = new Node(6);  
    Node f = new Node(5);  
    a.left = b;a.right = c;  
    b.left = d;b.right = e;  
    c.right = f;  
}
```

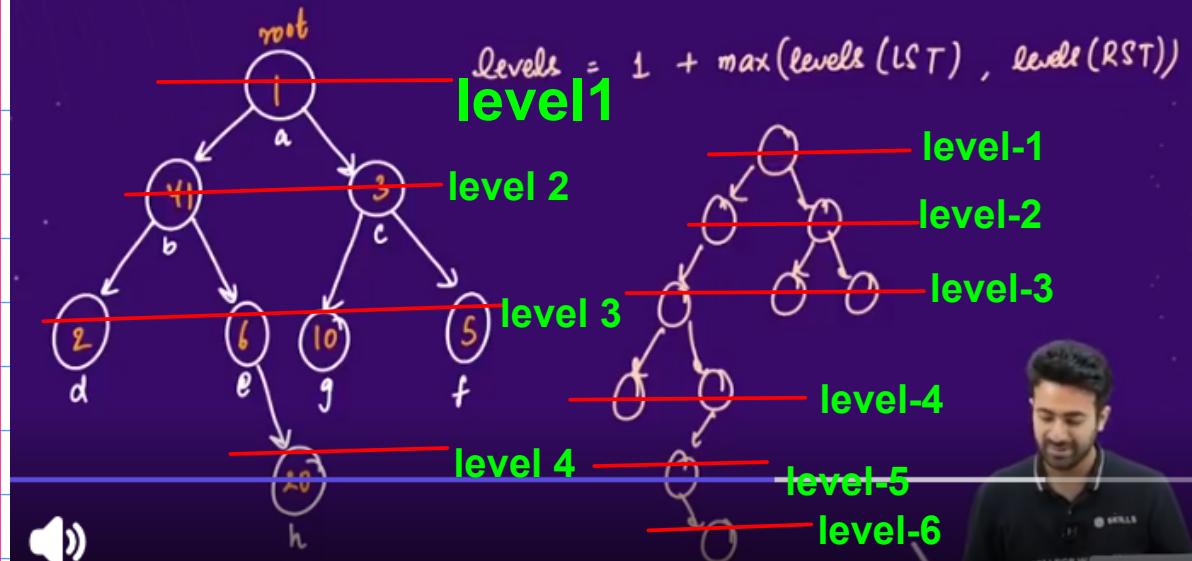
```
System.out.println( product(a));  
}
```

Find levels / height of Binary Tree



levels = 1 + max(levels(LST), levels(RST))





tree ki jo height hoti hai vo level se ek kam hota hai
ydi tree me height pucha hai to hm level nikalenge usme se 1 minus
kar denge to tree ki height ajayegi

```
private static int sum(Node root){
    if(root==null) return 0;
    return root.val + sum(root.left) + sum(root.right);
}

private static int size(Node root){
    if(root==null) return 0;
    return 1 + size(root.left) + size(root.right);
}

private static int levels(Node root){
    if(root==null) return 0;
    return 1 + Math.max(levels(root.left), levels(root.right));
}

private static int max(Node root){
    if(root==null) return Integer.MIN_VALUE;
    int a = root.val, b = max(root.left), c = max(root.right);
    return Math.max(a, Math.max(b, c));
}
```

treversal ka mtlb hai ki sabhi element per trevel karna hi traversal kahlata hai

```

private static void preorder(Node root){
    if(root==null) return;
    System.out.print(root.val+" ");
    preorder(root.left);
    preorder(root.right);
}

private static void inorder(Node root){
    if(root==null) return;
    inorder(root.left);
    System.out.print(root.val+" ");
    inorder(root.right);
}

private static void postorder(Node root){
    if(root==null) return;
    postorder(root.left);
    postorder(root.right);
    System.out.print(root.val+" ");
}

```

```

public static void main(String[] args) {
    Node a = new Node(1);           output
    Node b = new Node(2);
    Node c = new Node(3);
    Node d = new Node(4);
    Node e = new Node(5);
    Node f = new Node(6);
    Node g = new Node(7);
    a.left = b;a.right = c;
    b.left = d;b.right = e;
    c.left = f;c.right = g;
    display(a);
    System.out.println();
    pre(a);
    System.out.println();
    in(a);
    System.out.println();
    post(a);
    System.out.println();
}

```

pre,post and in order traversal

```

public class treetraversal {
    public static void display(Node root){
        if(root==null) return;
        System.out.print(root.val+" ");
        display(root.left);
        display(root.right);
    }

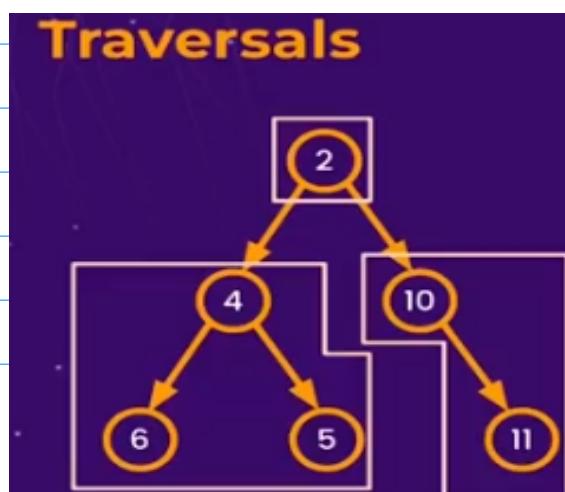
    public static void pre(Node root){
        if(root==null) return;
        System.out.print(root.val+" ");
        display(root.left);
        display(root.right);
    }

    public static void in(Node root){
        if(root==null) return;
        display(root.left);
        System.out.print(root.val+" ");
        display(root.right);
    }

    public static void post(Node root){
        if(root==null) return;
        display(root.left);
        display(root.right);
        System.out.print(root.val+" ");
    }
}

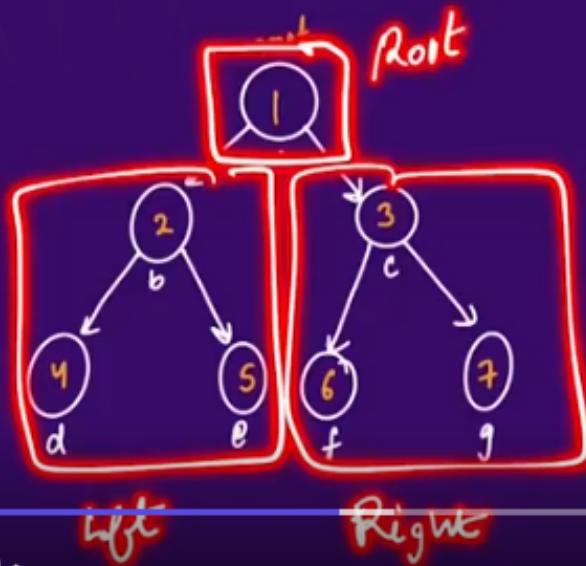
```

1 2 4 5 3 6 7 //display
 1 2 4 5 3 6 7 //pre order
 2 4 5 1 3 6 7 //in order
 2 4 5 3 6 7 1 //post order

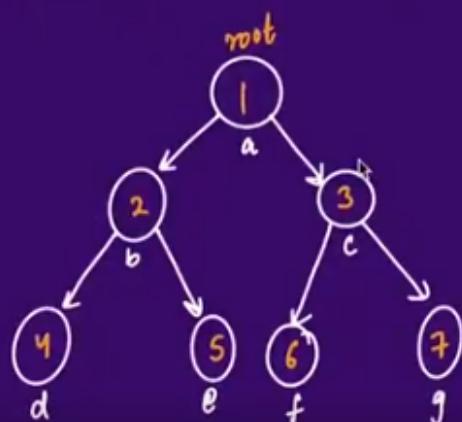


Root	L	R	PreOrder
L	Root	R	Inorder
L	R	Root	Postorder

Q : Binary Tree Preorder Traversal Root Left Right



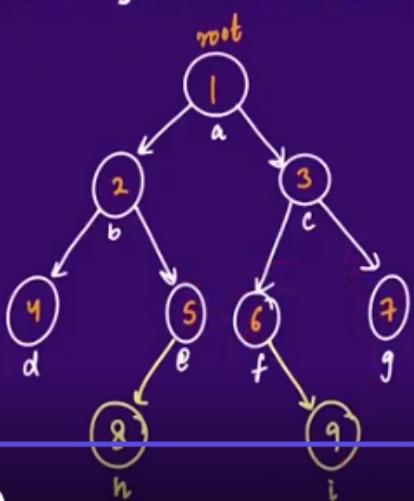
Q : Binary Tree Preorder Traversal Root Left Right



1 (2 5 4) (6 7 3)

1 — 2 4 5 — 3 6 7

Q : Binary Tree Preorder Traversal Root Left Right



1 (4 8 2 5) (3 6 7 9)

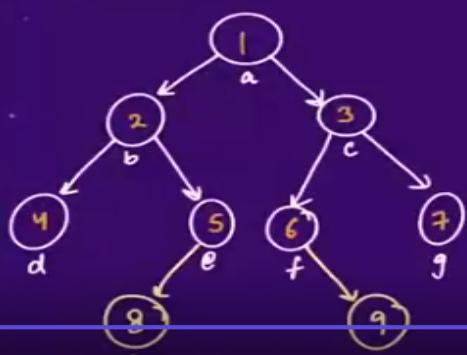
1 2 4 (8 5) 3 (6 9) 7

1 2 4 5 8 3 6 9 7



[Leet]

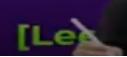
Q : Binary Tree Inorder Traversal Left Root Right



(2 4 5 8) 1 (3 6 7 9)

4 2 (5 8) 1 (6 9) 3 7

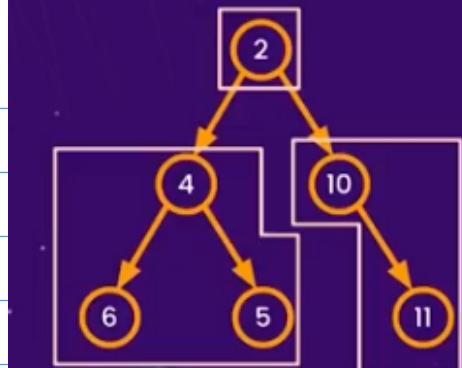
4 2 8 5 1 6 9 3 7



[Leet]

Traversals

Root L R PreOrder



L Root R Inorder

L R Root Postorder

Root R L Reverse PreOrder

R Root L Reverse Inorder

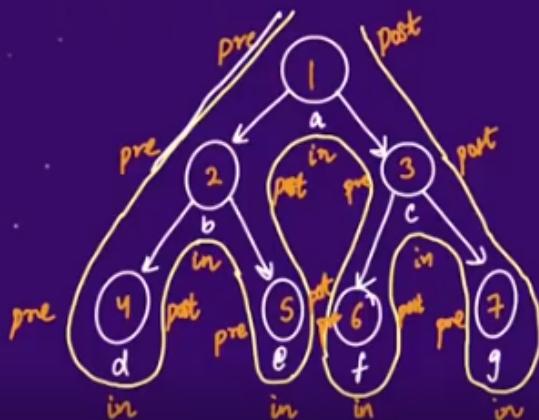
R L Root Reverse Postorder



Ques:

BFS traversals sk

Q : Binary Tree Preorder Traversal



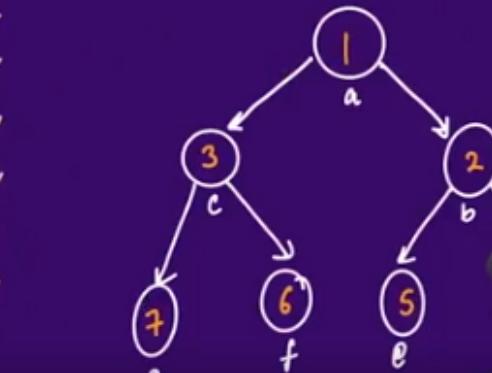
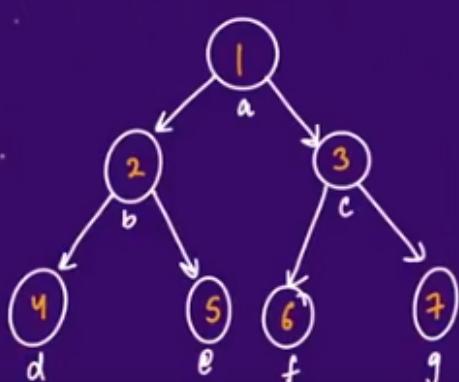
Pre: 1 2 4 5 3 6 7

In: 4 2 5 1 6 3 7

Post: 4 5 2 6 7 3 1

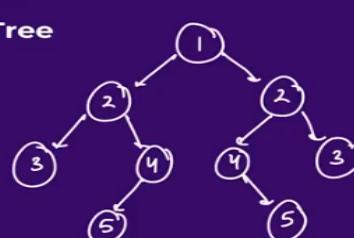


Q : Invert Binary Tree

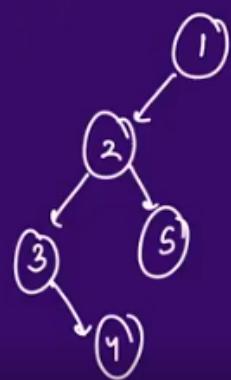


Homework:

Q : Symmetric Tree



Q : Diameter of Binary Tree



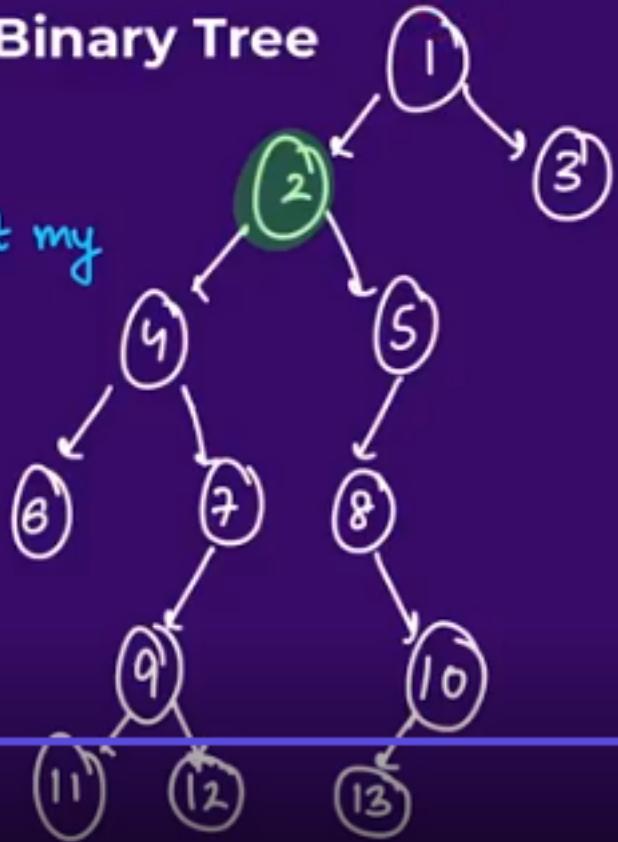
$$\text{diameter} = [\text{levels(LST)} + \text{levels(RST)} + 1] - 1$$



Q : Diameter of Binary Tree



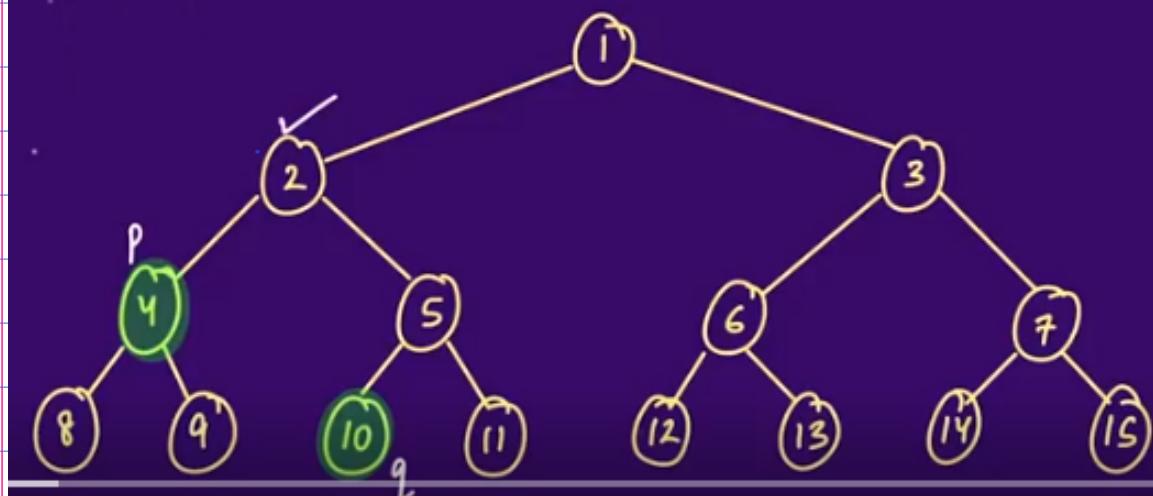
I was trying to get my diameter which is passing through the root.



Ques:

LCA is of ^{or more} 2 given nodes

Q : Lowest Common Ancestor of a Binary Tree



Description | Edit | Code

236. Lowest Common Ancestor of a Binary Tree

Medium Topics

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

Accepted Runtime: 16.1K

Test Result: Accepted

Case 1 Case 2

```

1 class Solution {
2     public boolean exists(TreeNode root, TreeNode node){
3         if(node==root) return true;
4         if(root==null) return false;
5         return exists(root.left,node) || exists(root.right,node);
6     }
7     public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
8         if(p==root || q==root) return root;
9         boolean pLiesInLST = exists(root.left,p);
10        boolean qLiesInLST = exists(root.left,q);
11        if(pLiesInLST==true && qLiesInLST==true) return lowestCommonAncestor(root.left,p,q);
12        if(pLiesInLST==false && qLiesInLST==false) return lowestCommonAncestor(root.right,p,q);
13        else return root;
14    }
15 }
```



236. Lowest Common Ancestor of a Binary Tree

Solved

Medium Topics Companies

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)."

```

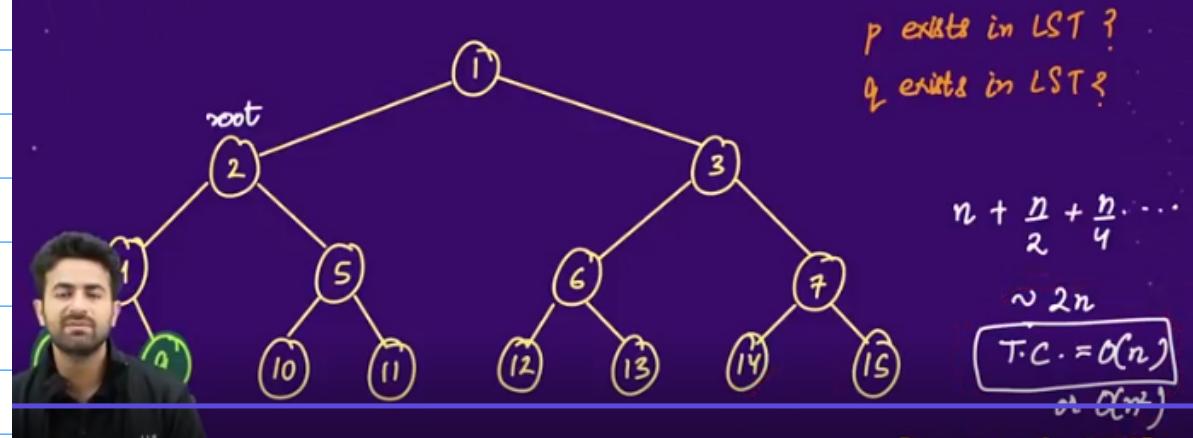
class Solution {
    public boolean exist(TreeNode root,TreeNode node){
        if(node==root) return true;
        if(root==null) return false;
        return exist(root.left,node)||exist(root.right,node);
    }
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
        if(p==root||q==root) return root;
        boolean pLieLST =exist(root.left,p);
        boolean qLieLST =exist(root.left,q);
        if(pLieLST==true&&qLieLST==true) return lowestCommonAncestor(root.left,p,q);
        if(pLieLST==false&&qLieLST==false) return lowestCommonAncestor(root.right,p,q);
        else return root;
    }
}
```

Ques:

Very Very Very Famous

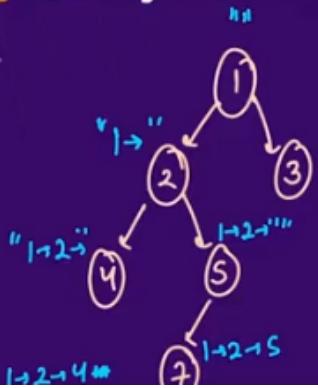
LCA is of ^{or more} 2 given nodes

Q : Lowest Common Ancestor of a Binary Tree



Time and space complexity

Q : Binary Tree Paths



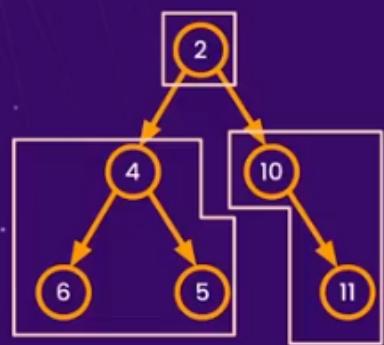
Root to Leaf path:

- 1) "1→2→4"
- 2) "1→2→5→7"
- 3) "1→3"

T.C. = $O(n)$
S.C. = $O(n \cdot h)$
height of tree

in all three case the time and space complexity

Traversals



T.C. = $O(n)$
S.C. = $O(n)$ → Recursion stack space

Root L R PreOrder



L Root R Inorder

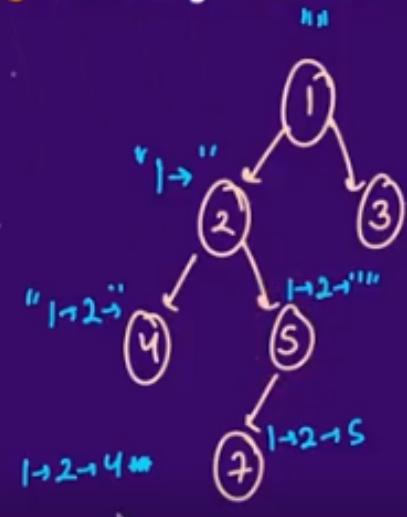
L R Root Postorder

Root R L Reverse PreOrder

R Root L Reverse Inorder

R L Root Reverse Postorder

Q : Binary Tree Paths



Root to Leaf path:

- 1) "1→2→4"
- 2) "1→2→5→7"
- 3) "1→3"

257. Binary Tree Paths

Solved

Easy

Topics

premium lock icon

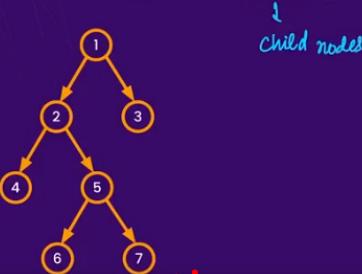
Companies

Given the root of a binary tree, return all root-to-leaf paths in any order.

A leaf is a node with no children.

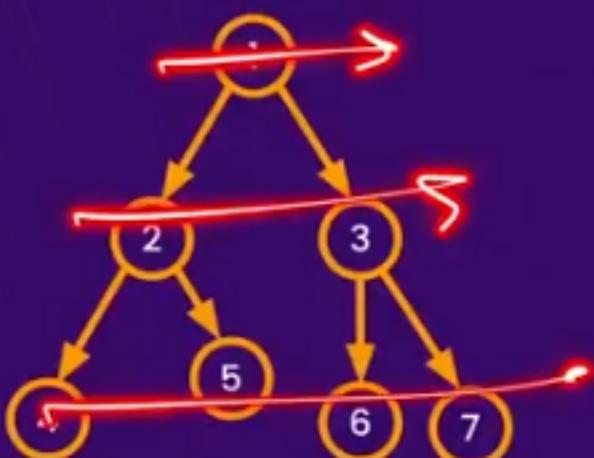
```
class Solution {  
    public void path(TreeNode root, String s, List<String> ans){  
        if(root==null) return;  
        if(root.left==null&&root.right==null){  
            s+=root.val;  
            ans.add(s);  
            return;  
        }  
        path(root.left, s+root.val+"->", ans);  
        path(root.right, s+root.val+"->", ans);  
    }  
    public List<String> binaryTreePaths(TreeNode root) {  
        List<String> ans = new ArrayList<>();  
        path(root, "", ans);  
        return ans;  
    }  
}
```

Full Binary Tree (0 or 2)



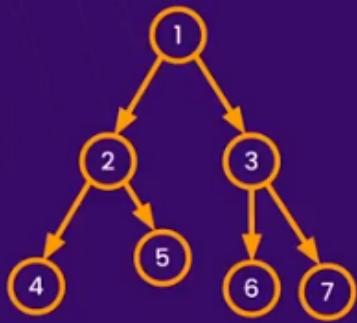
FULL BINARY TREE-->full binary tree vah tree hai jiske 0 ya to 2 child node ho to hm usse full binary tree kahte hai.

2. Perfect Binary Tree



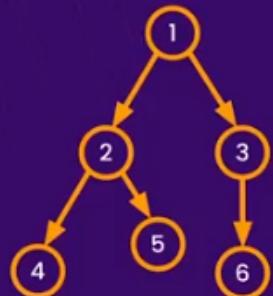
PERFECT BINARY TREE-->
perfect binary tree vah tree
hota hai jiske every node ke
two child hote hai.but leaf
node ke zero child hone
chahiye.

2. Perfect Binary Tree

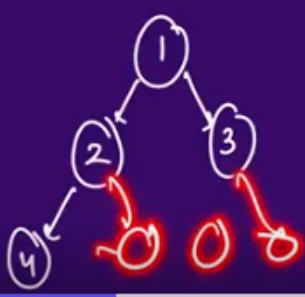


↓
Every node has 2 child nodes except the nodes of the level which have 0 child node

3. Complete Binary Tree



↓
Is a perfect binary tree but its last level can be incomplete



Yes

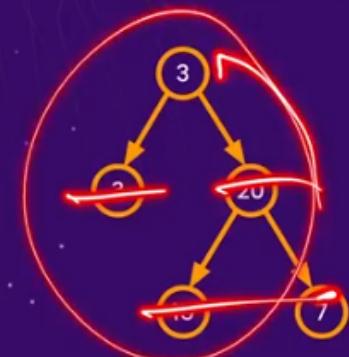
No



4. Balanced Binary Tree → Very Important

↓
where diff. b/w levels (LST) & levels (RST) ≤ 1

$$|\text{levels(LST)} - \text{levels(RST)}| \leq 1$$



In balanced Binary tree the levels of left subtree and levels of right subtree ka difference hai ≤ 1 hona chahiye tabhi vah balanced binary tree hogा. yah har ek root ke liye satisfy hona chahiye .

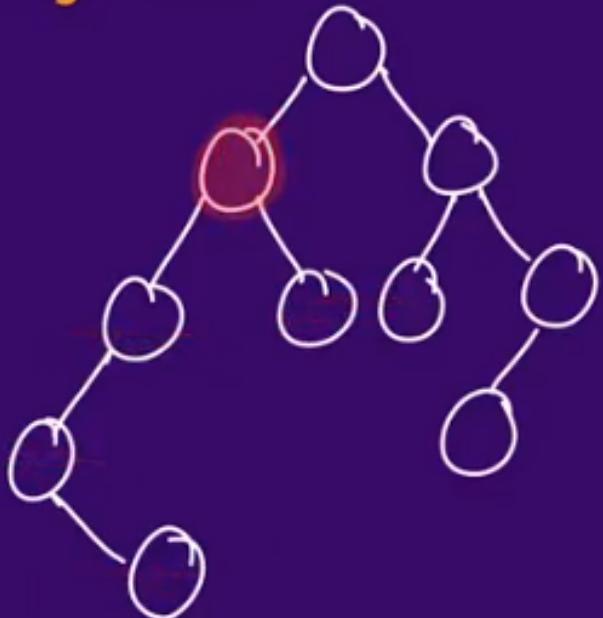
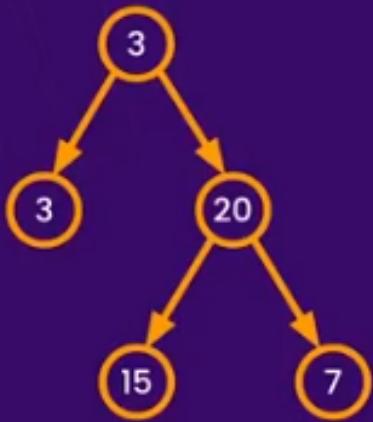
Balanced Binary Tree → Very Important

↓
where diff. b/w levels (LST) & levels (RST) ≤ 1

$|\text{levels(LST)} - \text{levels(RST)}| \leq 1$

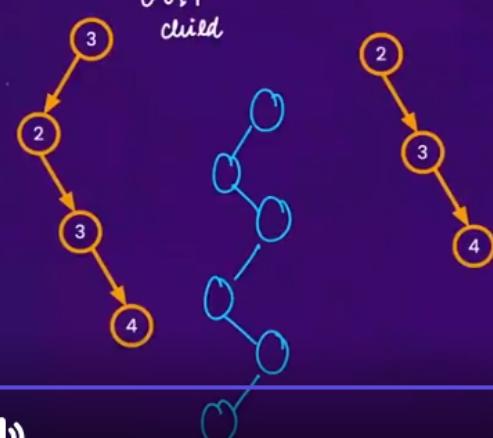
Unbalanced

4. Balanced Binary Tree



5. Degenerate and Skewed Binary Trees

0 or 1 child

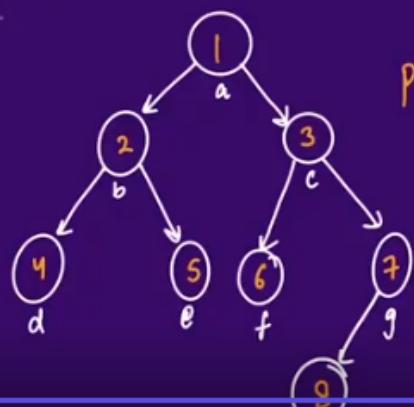


Degenerate binary tree-->
-->in this binary tree jiske
0 ya to 1 child hote hai for
every node that is called
Degenerate binary tree.

Skewed Binary trees-->
ye aisa binary tree hai jisme
ye to ya left me increase
hoga ya to keval right me hi
progress kargega like a line

Level order traversal (Using Queue)

BFS → breadth first search , level wise search



pop

push/add

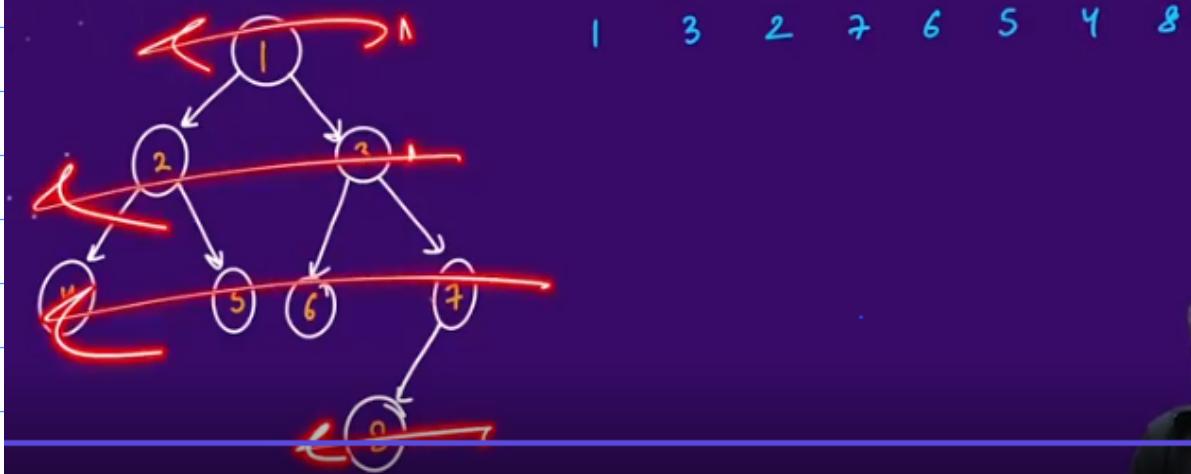
1 2 3 4 5 6 7 8

→ 1 2 3 4 5 6 7 8

Level order traversal (Using Queue)

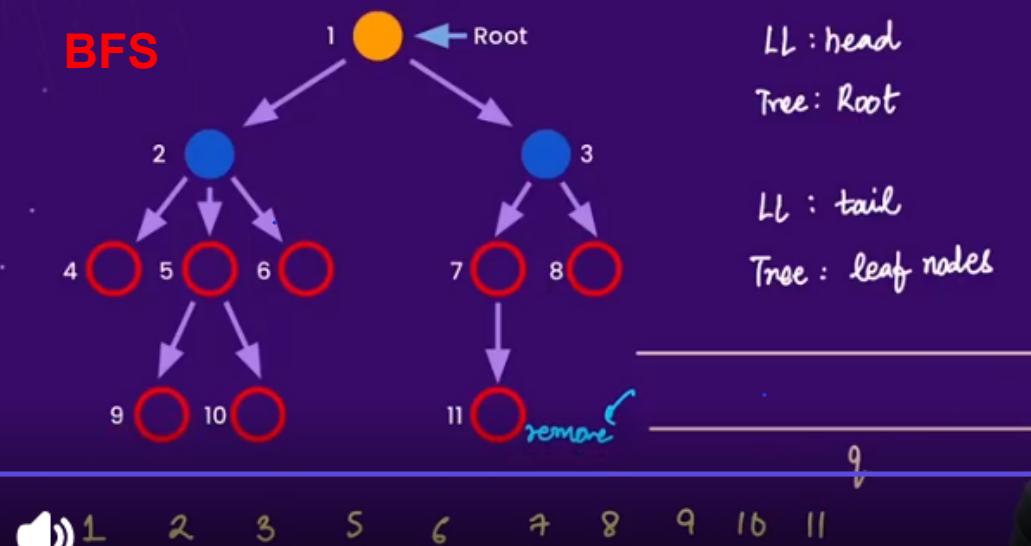
PW

Right to Left

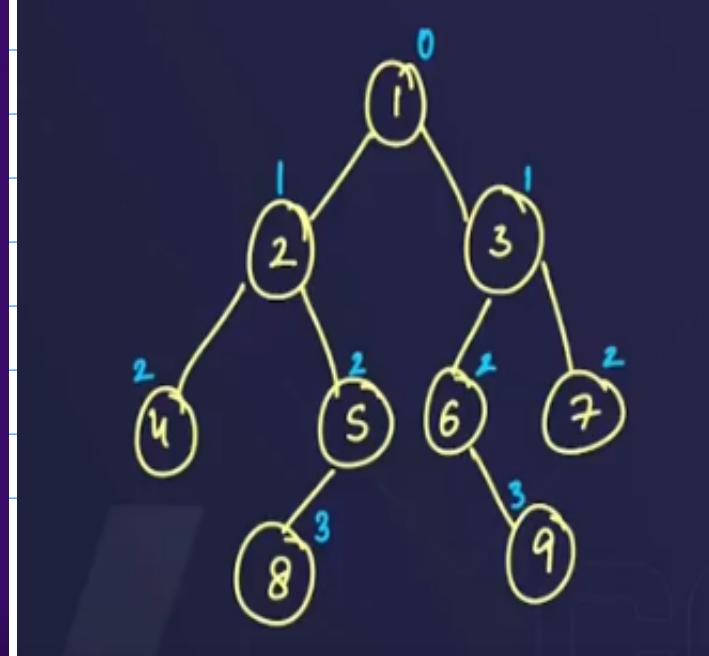
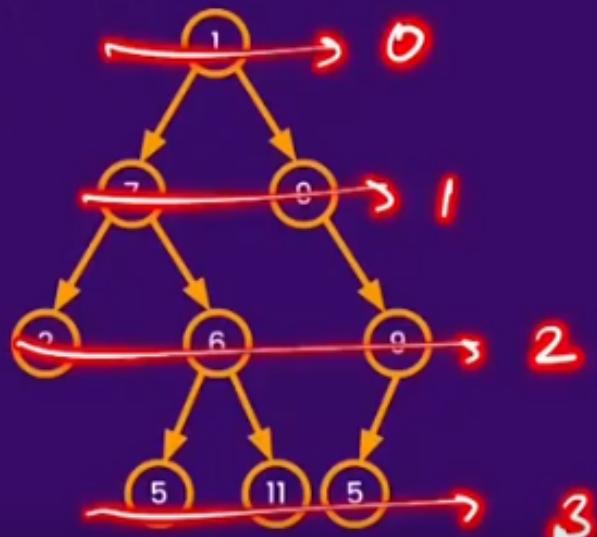


BFS--> means breadth first search jo hm queue se banate hai.

Representation



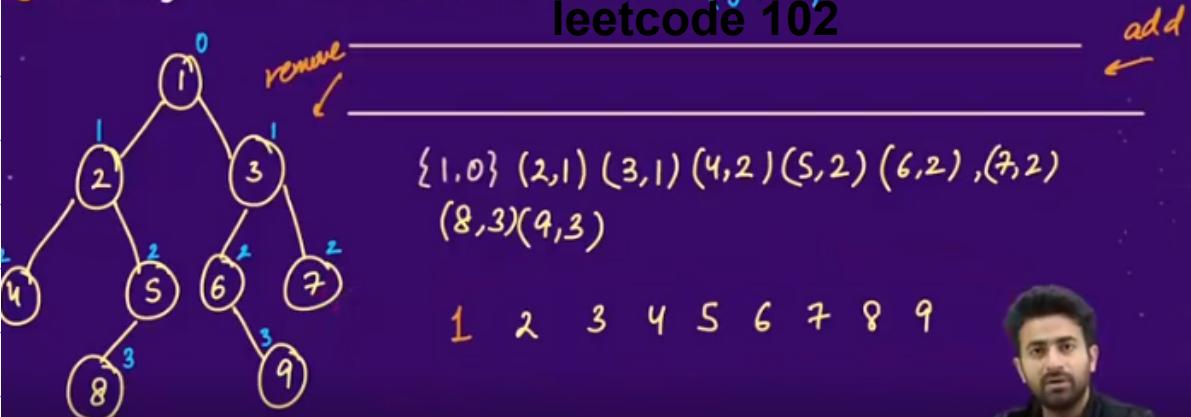
Print elements of nth level



Ques: Queue <Pair> $q = \text{new LinkedList}()$

Q : Binary Tree Level Order Traversal (Queue)

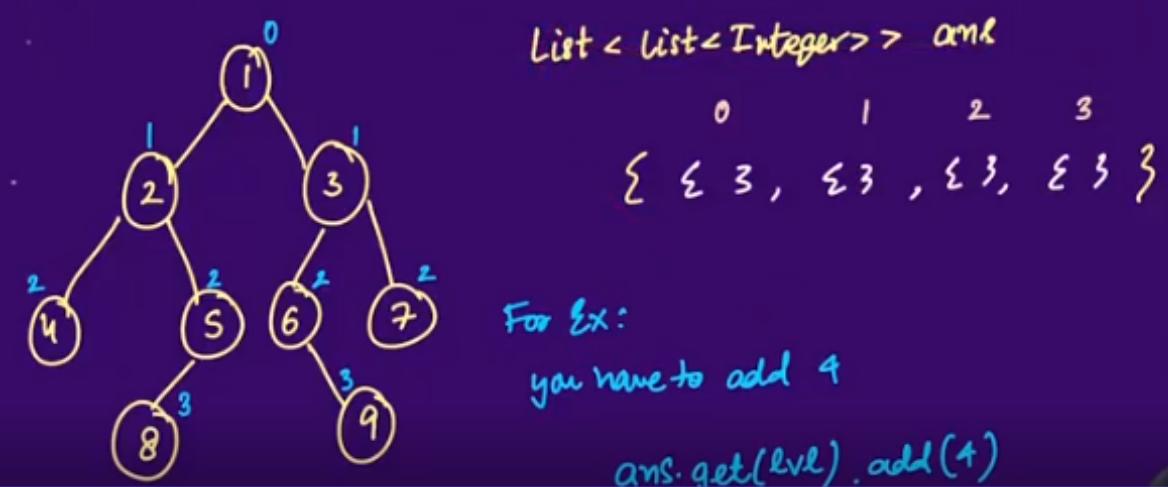
leetcode 102



Ques: T.C. = $O(n)$ S.C. = $O(n)$

Q : Binary Tree Level Order Traversal

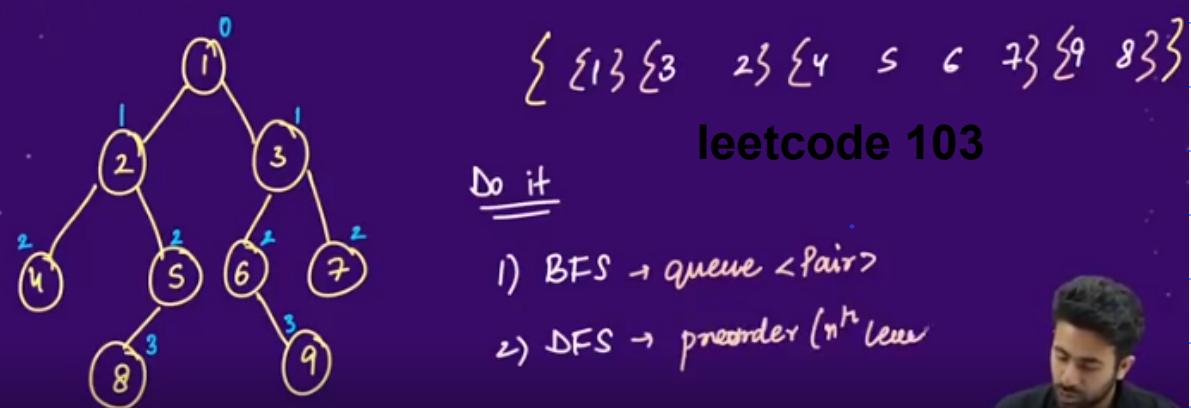
List < List < Integer > ans



Q : Binary Tree Level Order Traversal (ZigZag)

{ {1,3} {3,2} {4,5} {6,7} {9,8} }

leetcode 103



1. Preorder Traversal (Iterative)

Root Left Right

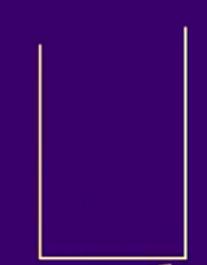
Recursive

↓

Stack

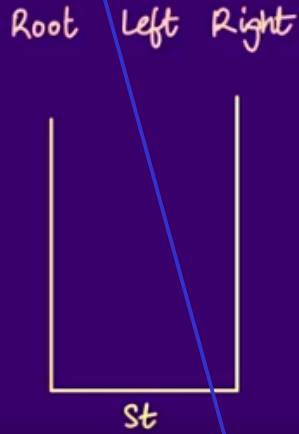
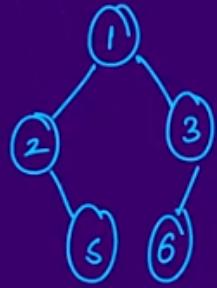
↓

Auxillary Space



this is iterative method

1. Preorder Traversal (Iterative)



Recursive ↓
Stack ↓
Auxillary Space



arr = { 1 2 5 3 6 7 }

144. Binary Tree Preorder Traversal

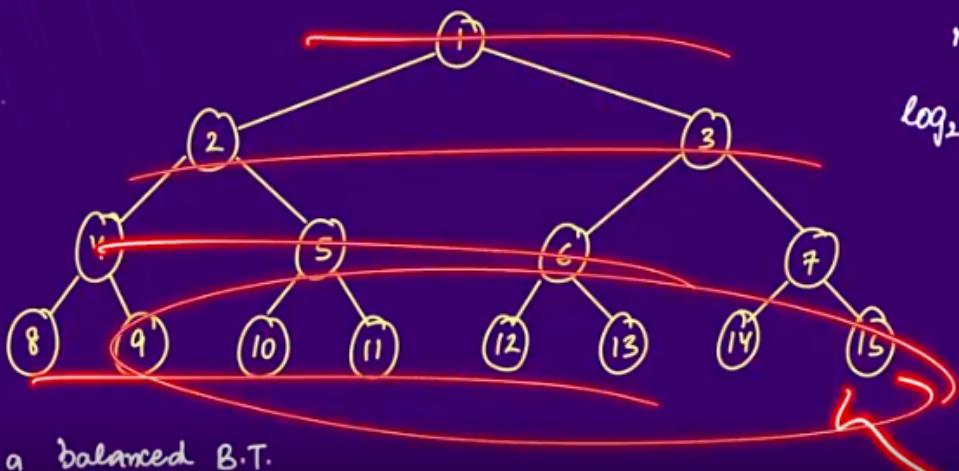
Solved

Easy Topics Companies

7.8K 63 Case 1 Case 2 Case 3

```
1 class Solution {
2     public List<Integer> preorderTraversal(TreeNode root) {
3         List<Integer> ans = new ArrayList<>();
4         Stack<TreeNode> st = new Stack<>();
5         if(root!=null) st.push(root);
6         while(st.size()!=0){
7             TreeNode top = st.pop();
8             ans.add(top.val);
9             if(top.right!=null) st.push(top.right);
10            if(top.left!=null) st.push(top.left);
11        }
12    }
13 }
14 }
```

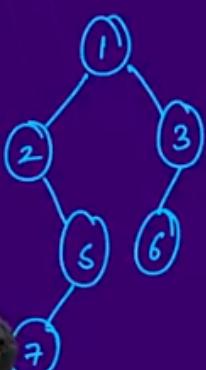
1. Preorder Traversal (Iterative)



For a balanced B.T.

if n are nodes then height/levels $\approx \log_2 n$

1. Preorder Traversal (Iterative)



Recursive ↓
Stack ↓
Auxillary Space

Time Complexity : $O(n)$

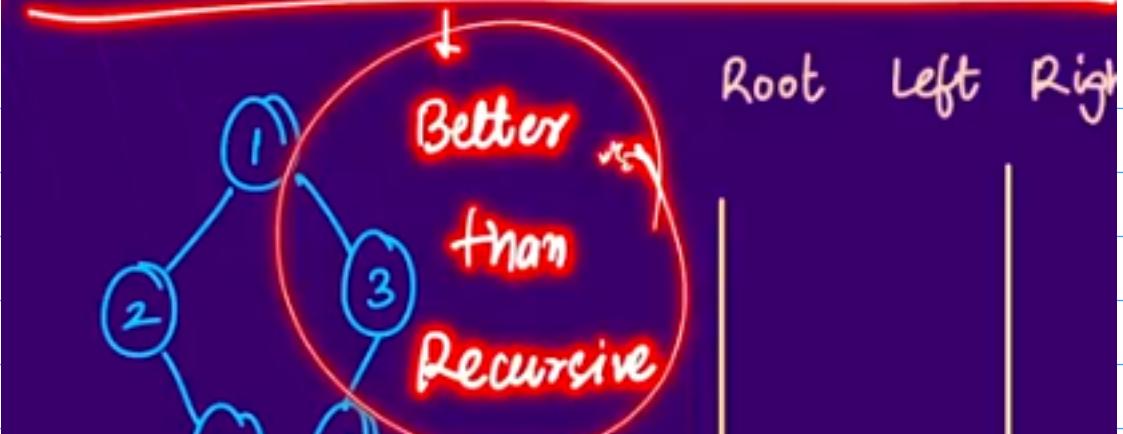
Space Complexity : $O(n)$

Auxillary Space : $O(h)$

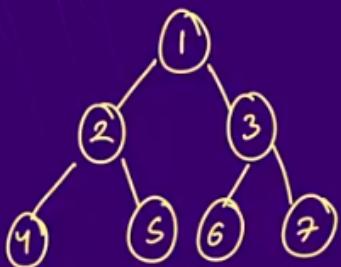
where h is level of tree

$h = \log n$

Preorder Traversal (Iterative)



3. Postorder Traversal (Iterative)



Left Right Root
4 5 2 6 7 3 1

→ postorder nothing but it is a reverse preorder

Reverse PreOrder

ans [1 3 7 6 2 5 4]
Kya hai ?



145. Binary Tree Postorder Traversal

Solved

Java Auto

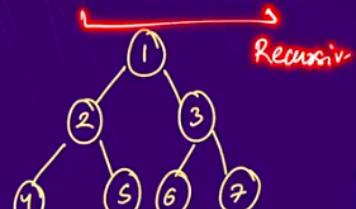
```
1 class Solution {
2     public List<Integer> postorderTraversal(TreeNode root) {
3         List<Integer> ans = new ArrayList<>(); //iterative method
4         Stack<TreeNode> st = new Stack<>();
5         if(root!=null) st.push(root);
6         while(st.size()>0){
7             TreeNode top = st.pop();
8             ans.add(top.val);
9             if(top.left!=null) st.push(top.left);
10            if(top.right!=null) st.push(top.right);
11        }
12        Collections.reverse(ans);
13        return ans;
14    }
15 }
```

Example 1:

Input: root = [1,null,2,3]

Output: [3,2,1]

3. Postorder Traversal (Iterative)



Left Right Root
4 5 2 6 7 3 1

Reverse (Root Right Left)
== PostOrder

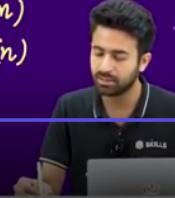
T.C. = O(n)

S.C. = O(n)

A.S. = O(n)



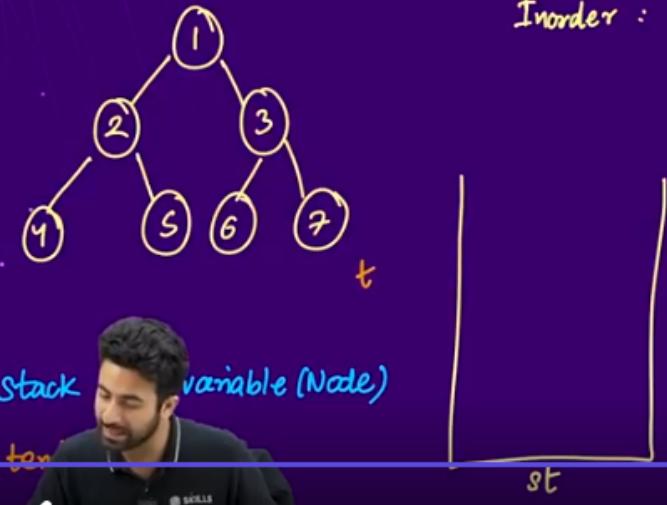
Reverse PreOrder
ans [1 3 7 6 2 5 4]
Kya hai ?



2. Inorder Traversal (Iterative)

Inorder : left root right

4 2 5 1 6 3 7



```
while(true){  
    if(temp!=null)  
        st.push(temp);  
        temp = temp.left;  
    else{  
        node_top = st.pop();  
        print  
        temp = top.right;  
        break  
    }  
}
```

94. Binary Tree Inorder Traversal

Solved

Easy Topics Companies

Given the `root` of a binary tree, return *the inorder traversal of its nodes' values.*

Example 1:

Input: `root = [1,null,2,3]`

Output: `[1,3,2]`

Explanation:

```
1 class Solution {  
2     public List<Integer> inorderTraversal(TreeNode root) {  
3         List<Integer> ans = new ArrayList<>();  
4         Stack<TreeNode> st = new Stack<>();  
5         TreeNode temp = root;  
6         while(true){  
7             if(temp!=null){  
8                 st.push(temp);  
9                 temp = temp.left;  
10            }  
11            else{  
12                if(st.size()==0) break;  
13                TreeNode top = st.pop();  
14                ans.add(top.val);  
15                temp = top.right;  
16            }  
17        }  
18    }  
19 }
```

2. Inorder Traversal (Iterative)

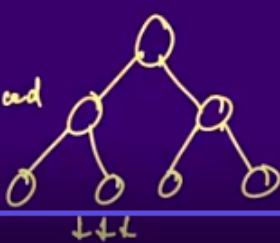
Time Complexity : $O(n)$

Space Complexity : $O(n)$

Auxiliary Space : Max Space of stack

best case

B.T.
 $\downarrow \downarrow L$



A.S $O(\log_2 n)$



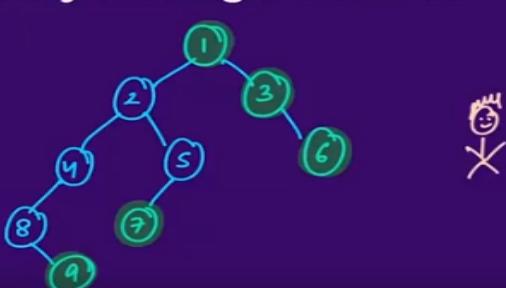
worst case

A.S = $O(n)$

auxillary space

Ques:

Q : **Binary Tree Right Side View** → All the nodes which are at the rightmost in each level



R.S = { 1, 3, 6, 7, 9 }



Ques:

Q : Binary Tree Right Side View



preorder = 1 2 4 8 9 5 7 3 6

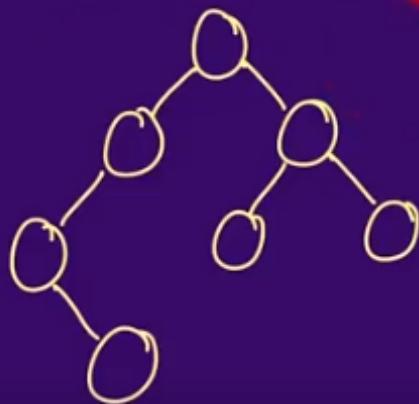
levels = 5

ans = {1, 3, 6, 7, 9}

inorder = 3 4 2 1 3 2 0 1 2



: Balanced Binary Tree



$$|\text{levels(LST)} - \text{levels(RST)}| \leq 1$$

true, false

leetcode 110



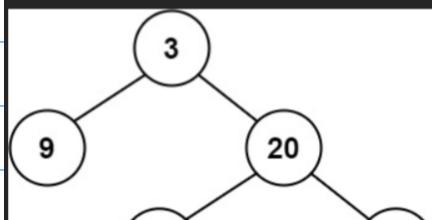
110. Balanced Binary Tree

Solved

[Easy](#) [Topics](#) [Companies](#)

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

Example 1:



Java Auto

```

1
2 class Solution {
3     public int levels(TreeNode root){
4         if(root==null) return 0;
5         return 1+Math.max(levels(root.left),levels(root.right));
6     }
7     public boolean isBalanced(TreeNode root) {
8         if(root==null) return true;
9         int diff = Math.abs(levels(root.left)-levels(root.right));
10        if(diff>1) return false;
11        boolean lbt =isBalanced(root.left);
12        if(lbt==false) return false;
13        boolean rst =isBalanced(root.right);
14        if(rst==false) return false;
15        return true;
16    }
17 }
```

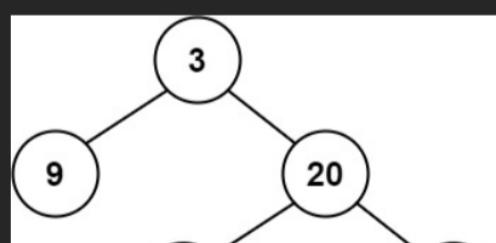
110. Balanced Binary Tree

Solved

[Easy](#) [Topics](#) [Companies](#)

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

Example 1:



Java Auto

```

4     if (root == null)
5         return 0;
6         return 1 + Math.max(levels(root.left), levels(root.right));
7     }
8
9     public boolean isBalanced(TreeNode root) {
10        if (root == null)
11            return true;
12        int diff = Math.abs(levels(root.left) - levels(root.right));
13        if (diff > 1)
14            return false;
15        if (!isBalanced(root.left))
16            return false;
17        if (!isBalanced(root.right))
18            return false;
19        return true;
20    }
21 }
```

110. Balanced Binary Tree

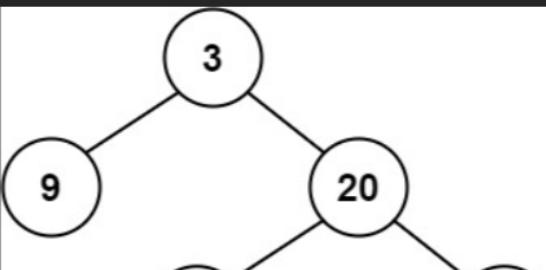
Solved



Easy Topics Companies

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

Example 1:



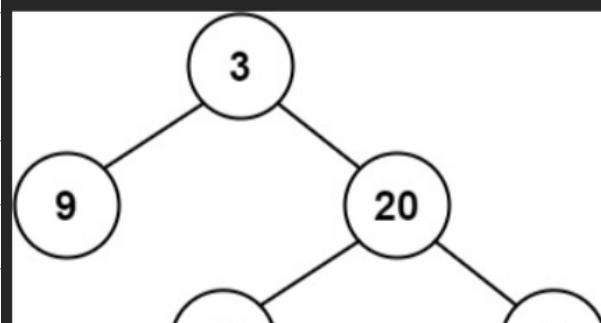
110. Balanced Binary Tree

Solved

Easy Topics Companies

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

Example 1:



```
1
2 class Solution {
3     public int levels(TreeNode root) {
4         if (root == null)
5             return 0;
6         return 1 + Math.max(levels(root.left), levels(root.right));
7     }
8
9     public boolean isBalanced(TreeNode root) {
10        if (root == null)
11            return true;
12        int diff = Math.abs(levels(root.left) - levels(root.right));
13        if (diff > 1) return false;
14        return isBalanced(root.left)&&isBalanced(root.right);
15    }
16 }
```

Java Auto

```
2 class Solution {
3     static boolean ans;
4     public int levels(TreeNode root) {
5         if (root == null) return 0;
6         int leftlevel = levels(root.left);
7         int rightlevel = levels(root.right);
8         int diff = Math.abs(leftlevel-rightlevel);
9         if(diff>1) ans= false;
10        return 1 + Math.max(leftlevel,rightlevel );
11    }
12
13    public boolean isBalanced(TreeNode root) {
14        ans=true;
15        levels(root);
16        return ans;
17    }
18 }
```

optimise code

Ques:

Q : Diameter of Binary Tree



I was trying to get my diameter which is passing through the root.



dia = 6
dia = 8

SKILLS

dia = {6, 8, 0, 4, 0, 2, 2, 0, 0, 3, 2, 1, 0}

T.C. = O(n²)

S.C. = O(n)



543. Diameter of Binary Tree

Solved

Tree

Easy Topics Companies

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.

```
1 class Solution {  
2     static int maxDia;  
3     public int levels(TreeNode root){  
4         if(root==null) return 0;  
5         int leftlevel = levels(root.left);  
6         int rightlevel = levels(root.right);  
7         int mydia = leftlevel+rightlevel;  
8         maxDia = Math.max(mydia,maxDia);  
9         return 1+Math.max(leftlevel,rightlevel);  
10    }  
11    public int diameterOfBinaryTree(TreeNode root) {  
12        maxDia= 0;  
13        levels(root);  
14        return maxDia;  
15    }  
16 }  
17 }  
18 }
```

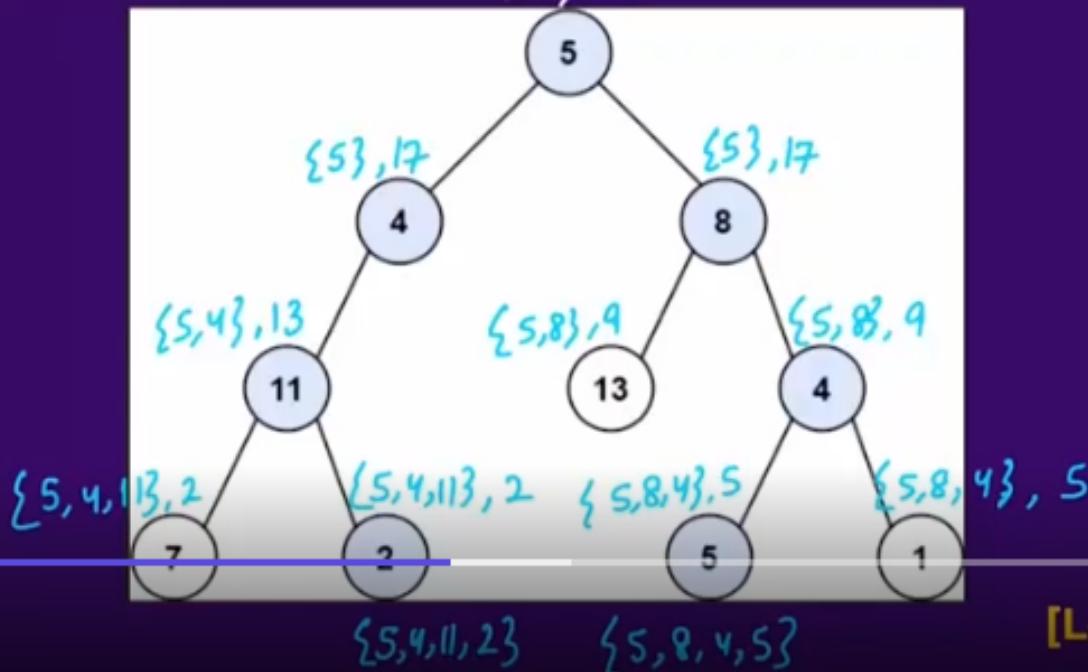
Ques: $\text{ans} = \{\{5, 4, 11, 23\}, \{5, 8, 4, 53\}\}$



Q : Path Sum II

$\{3, 22\}$

$\text{target} = 22$



[Lea

Solved

113. Path Sum II

Medium Topics Companies

Given the `root` of a binary tree and an integer `targetSum`, return all **root-to-leaf** paths where the sum of the node values in the path equals `targetSum`. Each path should be returned as a list of the node **values**, not node references.

A **root-to-leaf** path is a path starting from the root and ending at any leaf node. A **leaf** is a node with no children.

```

class Solution {
    public List<Integer> copy(List<Integer> arr) {
        List<Integer> list = new ArrayList<>();
        for (int ele : arr) {
            list.add(ele);
        }
        return list;
    }

    public void helper(TreeNode root, int target, List<Integer> arr, List<List<Integer>> ans) {
        if (root == null)
            return;
        if (root.left == null && root.right == null) {
            if (root.val == target) {
                arr.add(root.val);
                ans.add(arr);
            }
            return;
        }
        arr.add(root.val);
        List<Integer> arr1 = copy(arr);
        List<Integer> arr2 = copy(arr);
        helper(root.left, target - root.val, arr1, ans);
        helper(root.right, target - root.val, arr2, ans);
    }

    public List<List<Integer>> pathSum(TreeNode root, int targetSum) {
        List<List<Integer>> ans = new ArrayList<>();
        List<Integer> arr = new ArrayList<>();
        helper(root, targetSum, arr, ans);
        return ans;
    }
}

```

Q : Path Sum III

```

public int helper(TreeNode root, long sum){
    if(root==null) return 0;
    int count = 0;
    if(root.val==sum) count++;
    count += helper(root.left,sum-root.val) + helper(root.right,sum-root.val);
    return count;
}
public int pathSum(TreeNode root, int sum) {
    if(root==null) return 0;
    return helper(root,sum) + pathSum(root.left,sum) + pathSum(root.right,sum);
}

```

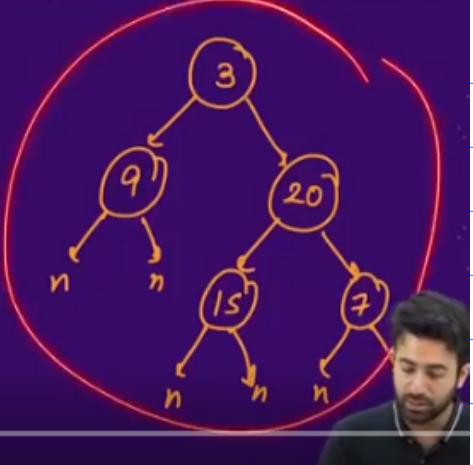
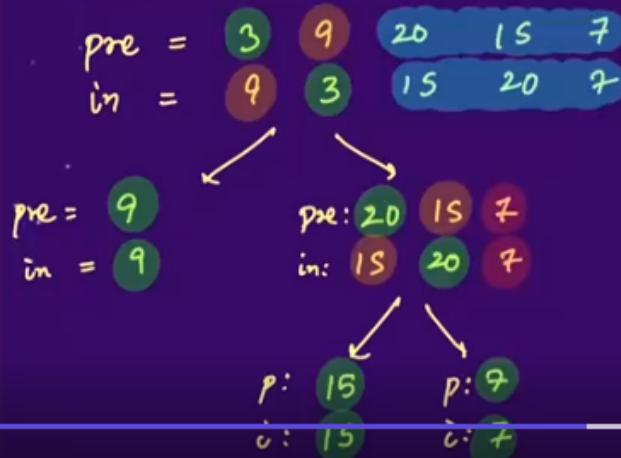
Q : Construct Binary Tree from Preorder & Inorder Traversal



[Lee]

Ques: Think about root of the tree

Q: Construct Binary Tree from Preorder & Inorder Traversal



[Leet]

Q: Construct Binary Tree from Preorder & Inorder Traversal



(3)

LST size = r - il

LST → pl+1 to pl+ls , il to r-1

RST → pl+ls+1 to ph , r+1 to ih

[Lee]



Assignment Questions

Q. Level Order Traversal (Right to Left).

Q. Find the product of all nodes in a Binary Tree.

Q. ZigZag level order traversal (Leetcode 103).

Q. Find the minimum value in a Binary tree.

Q. Construct Binary Tree from Inorder & Postorder Traversal (Leetcode 106).

Q. Balanced Binary Tree (Leetcode 110).

Q. Construct Binary Tree from Preorder & Postorder Traversal (Leetcode 889).

Q. Symmetric Tree (Leetcode 101)

Q. Left View of Binary Tree

Given a Binary Tree, print the Left view of it.

The left view of a binary tree refers to the set of nodes that are visible when the tree is viewed from the left side.

Q. Path Sum I

[LeetCode 112]