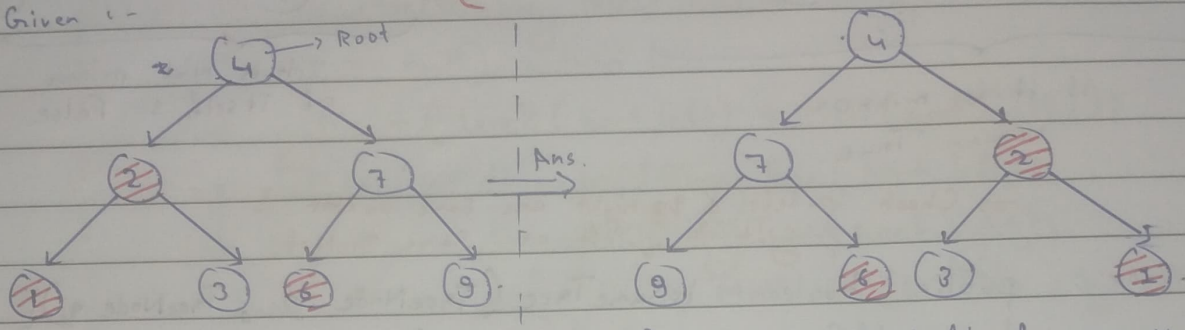
# Trees [LeetCode Question]

M T W T F S S | Page No.: | Date: | YOUVA

=> LeetCode Q. No. (144), (94), (145) are in Lecture Notes.

① LeetCode Q. No. (226) { Invert Binary Tree }

Given :-

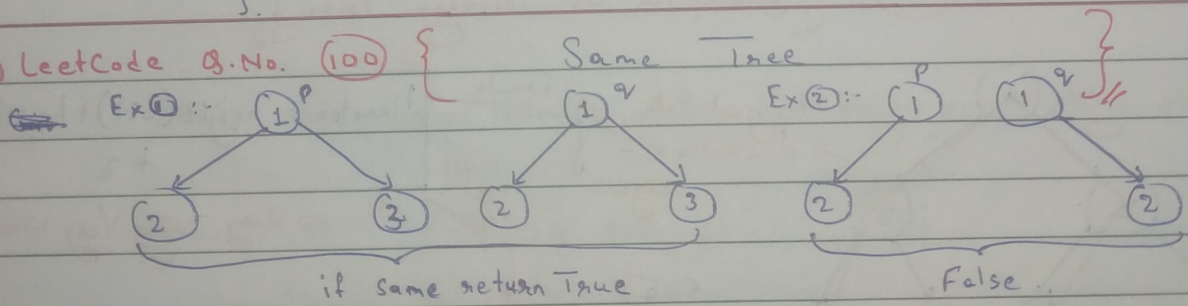



Logic :- Chang root.left to right & root.right to left. for every Node

=> Code :-

```
public TreeNode invertTree(TreeNode root){
    if(root == null) return null;
    TreeNode temp = root.left;
    root.left = root.right;
    root.right = temp;
    invertTree(root.left);
    invertTree(root.Right);
    return root;
}
```

② LeetCode Q. No. (100) { Same Tree }

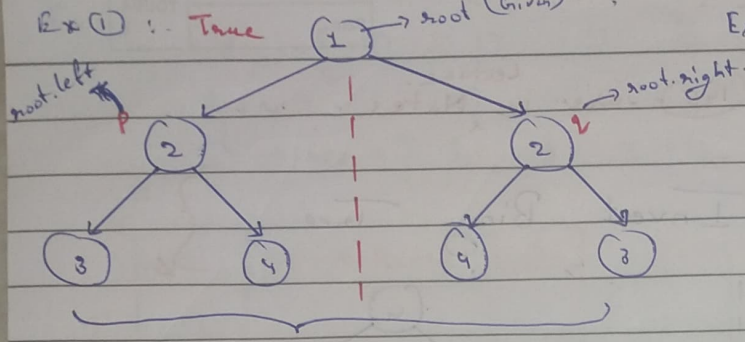



=> Code :-

```
public boolean isSameTree(TreeNode p, TreeNode q){
    if(p == null && q == null) return true;
    if(p == null || q == null) return false;
    if(p.val != q.val) return false;
    if(!isSameTree(p.left, q.left)) return false;
    if(!isSameTree(p.right, q.right)) return false;
    return true;
}
```

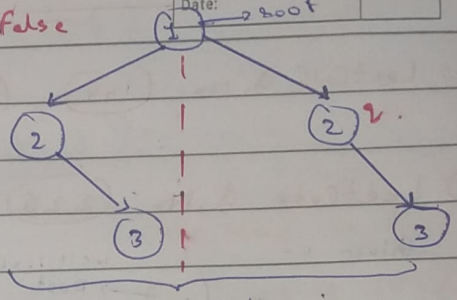
(3) LeetCode Q.No. :- (101) { Symmetric Tree (Given) } //

Ex (1) :- True




if it is mirror return True

Ex (2) False




It is Not mirror of itself so False

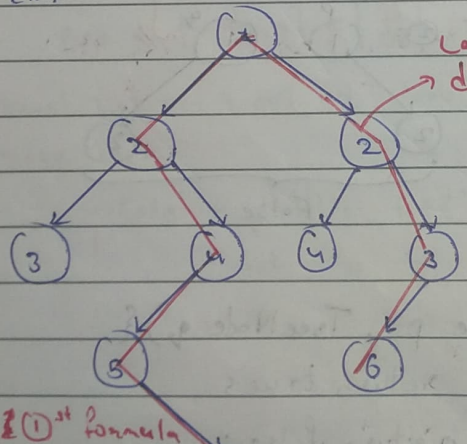
=> Check p.left & q.right are same or Not & p.right & q.left are same or Not.

=> Code :-

```
public boolean isSameTree (TreeNode p , TreeNode q){
    if(p == null && q == null) return true;
    if(p == null || q == null) return false;
    if(p.val != q.val) return false;
    if(!isSameTree(p.left , q.right)) return false;
    if(!isSameTree(p.right , q.left)) return false;
    return true;
}

public boolean isSymmetric(TreeNode root){
    return isSameTree(root.left , root.right);
}
```

(4) LeetCode Q.No. (543) { Diameter (Largest Path) of Binary Tree } //

Ex (1) :-

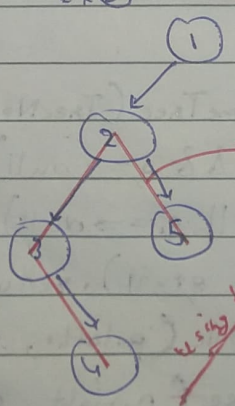



Largest Path diameter = 7 edges.

Formula (1) → diameter = height(LST) + height(RST) + 2

Using this formula → Very Very Wrong.

Ex (2) :-




diameter = 2 + 0 + 2 => 4 ✗

Largest diameter => 3 //.

diameter => 2 + 1 => 3 //

Using this

In (1)st formula I was trying to get my diameter which is passing through the root, but this is not correct for all Trees (Ex 2).

(2) New Formula :-

[ diameter => levels(LST) + levels(RST) ] //

⇒ Code :-

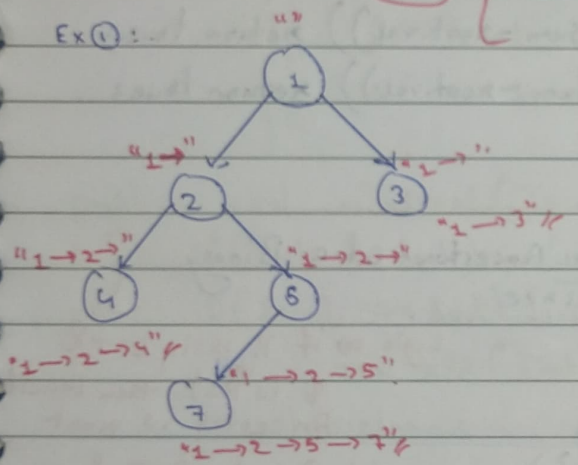
T.C ⇒ $O(n^2)$ // S.C ⇒ $O(n)$ ↓ rec.

```
public int levels(TreeNode root){
    if(root == null) return 0;
    int level = 1 + Math.max(levels(root.left), levels(root.right));
    return level;
}

public int diameterOfBinaryTree(TreeNode root){
    if(root == null) return 0;
    int myDia = (levels(root.left) + levels(root.right));
    int leftDia = diameterOfBinaryTree(root.left);
    int rightDia = diameterOfBinaryTree(root.right);
    return Math.max(myDia, Math.max(leftDia, rightDia));
}
```

---

⑤ Leet Code Q.No:- (257) { Binary Tree Paths }

Ex ①:




# Root to Leaf Paths:-

① " 1 ⟶ 2 ⟶ 4"

② " 1 ⟶ 2 ⟶ 5 ⟶ 7"

③ " 1 ⟶ 3"

# T.C ⇒ O(n)

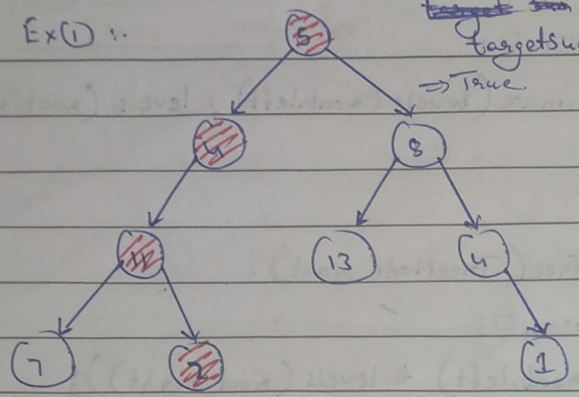
# S.C ⇒ O(n.h) ↓ height of tree

⇒ Code :-

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

⑥ LeetCode Q.No. (112) { Path Sum }

Ex ① :-

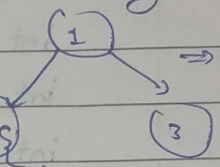



~~target~~ ~~Sum~~ targetSum = 22

⇒ True

Any root to leaf path sum of Nodes.val is equal to targetSum then return True else False.

Ex ②.



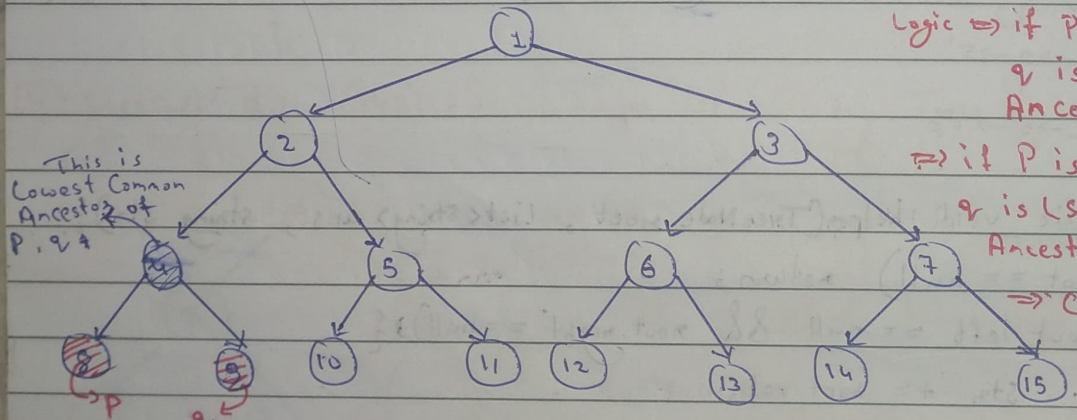

targetSum = 5

⇒ False.

⇒ Code :-

```
public boolean hasPathSum(TreeNode root, int targetSum) {
    if(root == null) return false;
    if(root.left == null && root.right == null){
        if(targetSum - root.val == 0) return true;
    }
    if(hasPathSum(root.left, targetSum - root.val)) return true;
    if(hasPathSum(root.right, targetSum - root.val)) return true;
    return false;
}
```

⑦ LeetCode Q.No. (236) { Lowest Common Ancestor of a Binary Tree. }




Logic ⇒ if p is in LST & q is in RST then Lowest Ancestor is root.

⇒ if p is in RST & q is LST then Lowest Ancestor is root

⇒ Check for every Node.

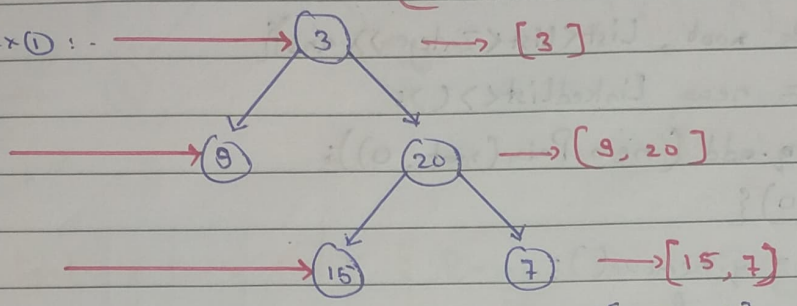
⇒ Code :-

```
public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
    if(p == root || q == root) return root;
    boolean pliesInLST = ifExist(root.left, p);
    boolean qliesInLST = ifExist(root.left, q);
    if(pliesInLST && qliesInLST) return lowestCommonAncestor(root.left,p,q);
    if(!pliesInLST && !qliesInLST) return lowestCommonAncestor(root.right,p,q);
    else return root;
}
```

```
public boolean ifExist(TreeNode root, TreeNode x){
    if(root == null) return false;
    if(root == x) return true;
    return ifExist(root.right, x) || ifExist(root.left, x);
}
```

(8) LeetCode Q.No. 102 { Binary Tree Level Order Traversal }

Ex①:-




# Using Nth Level Method. → [Method - 1]

=> Code :-

```
public int levels(TreeNode root){
    if(root == null) return 0;
    int level = 1 + Math.max(levels(root.left), levels(root.right));
    return level;
}

public void nThLevel(Tree Node root, int level, int lvl, List<Integer> arr){
    if(root == null) return;
    if(level == lvl){
        arr.add(root.val);
        return;
    }
    nThLevel(root.left, level+1, lvl, arr);
    nThLevel(root.right, level+1, lvl, arr);
}

public List<List<Integer>> levelOrder(TreeNode root){
    List<List<Integer>> ans = new ArrayList<>();
    int lvl = levels(root);
    for(int i=0; i<lvl; i++){
        List<Integer> arr = new Arraylist<>();
        nThLevel(root, 0, i, arr);
        ans.add(arr);
    }
    return ans;
}
```

# Using ~~level~~ (Queue) ——→ (Method - 2)

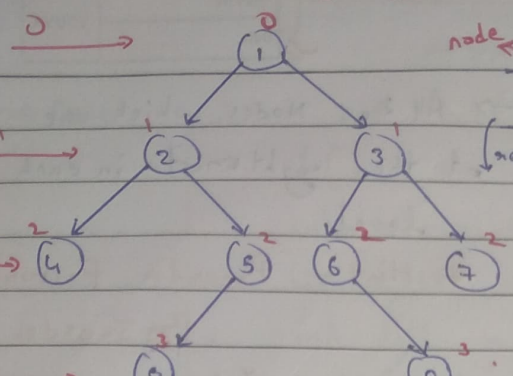
M T W T F S S | Page No.: | Date: | YOUVA

=> Code :-

```
public int levels(TreeNode root){
    if (root == null) return 0;
    int level = 1 + Math.max(levels(root.left), levels(root.right));
    return level;
}

public void helper(TreeNode root, List<List<Integer>> ans){
    Queue<Pair> q = new LinkedList<>();
    if(root != null) q.add(new Pair(root, 0));
    while(q.size() > 0){
        Pair front = q.remove();
        TreeNode temp = front.node;
        int lvl = front.level;
        ans.get(lvl).add(temp.val);
        if(temp.left != null) q.add(new Pair(temp.left, lvl+1));
        if(temp.right != null) q.add(new Pair(temp.right, lvl+1));
    }
}

public List<List<Integer>> levelOrder(TreeNode root){
    List<List<Integer>> ans = new ArrayList<>();
    int lvl = levels(root);

    for(int i=0; i<=lvl; i++){
        List<Integer> arr = new ArrayList<>();
        ans.add(arr);
    }

    helper(root, ans);
    return ans;
}

public Class Pair {
    TreeNode node;
    int level;
    Pair(TreeNode node, int level){
        this.node = node;
        this.level = level;
    }
}
```

Explanation :- Queue<Pair> q = new LinkedList<>();

Pair → node, lvl




node, lvl.

{1,0}, {2,1}, {3,1}, {4,2}, {5,2}, {6,2}, {7,2}, {8,3}, {9,3}

remove — Add

List<List<Integer>> ans;

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

For Ex :- you need to Add (4).

ans.get(lvl).add(4);

(9) LeetCode Q.No. (103) { Binary Tree Zigzag Level Order Traversal. } → Same to Previous one need to make Little Changes to Code

Take this Tree for Ex (1)

→ Output :- { {1}, {3, 2}, {4, 5, 6, 7}, {9, 8} }

* Using Mth Level :- (M-1)

```
=> Code :- public void nThLevel(TreeNode root, int level, int lvl, List<Integer> arr){
    if(root == null) return;
    if(level == lvl){
        arr.add(root.val);
        return;
    }
    if(lvl % 2 == 0){
        nThLevel(root.left, level+1, lvl, arr);
        nThLevel(root.right, level+1, lvl, arr);
    }
    else{
        nThLevel(root.right, level+1, lvl, arr);
        nThLevel(root.left, level+1, lvl, arr);
    }
}
```

# Only need to modify nThLevel method little bit.

Changes.

* Using [Queue] → (M-2) → Just reverse the Arr which are in odd indexes.

```
=> Code :- p v reverseArr(List<Integer> arr){
    int i = 0, j = arr.size()-1;
    while(i<j){
        int temp = arr.get(i);
        arr.set(i, arr.get(j));
        arr.set(j, temp);
        i++; j--;
    }
}
```

```
for(int i=0; i<lvl; i++){
    if(i%2 != 0){
        reverseArr(ans.get(i));
    }
}
```

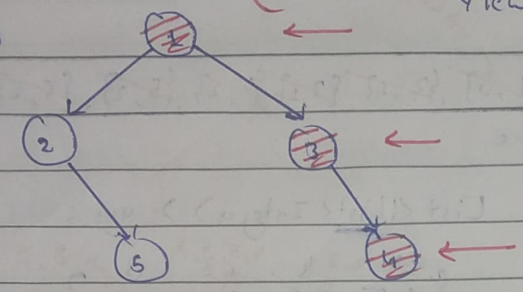
Add this after the helper method call

→ Add in levelOrder Method

(10) Leet Code Q. No. (199) { Binary Tree Right Side View.

M T W T F S S | Page No.: | Date: | YOUVA

Ex (1)



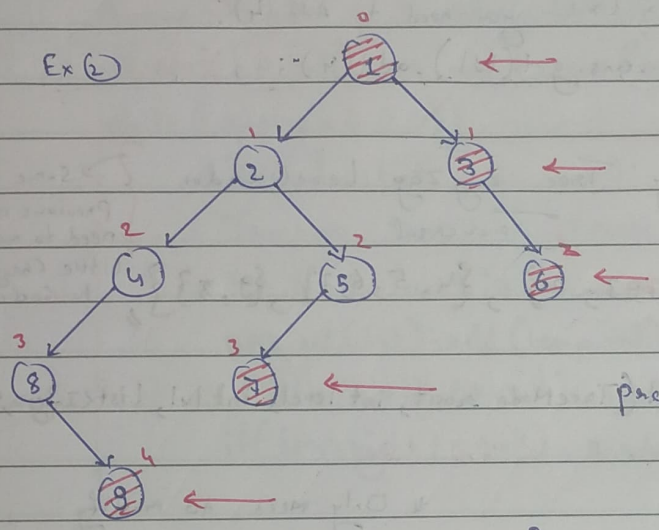
=> All the Nodes which are at the rightmost in each level.

# Hint :- Preorder, Postorder, ~~Post~~ Inorder.

$n^{th}$ Level.

Ans => {1, 3, 4}

Ex (2)



Dry Run.

Level => 5

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| ans { | 1 | ~~2~~ 3 | ~~4~~ 6 | ~~8~~ 7 | 9 } |

preorder = 1 2 4 8 9 5 7 3 6

0 1 2 3 4 2 3 1 2

Ans => {1, 3, 6, 7, 9}

=> Code :-

```
public int lvl(TreeNode root){
    if(root == null) return 0;
    return 1 + Math.max(lvl(root.right), lvl(root.left));
}

public void dfs(TN root, int level, List<Integer> ans){
    if(root == null) return;
    dfs(root.left, level+1, ans);
    dfs(root.right, level+1, ans);
    ans.set(level, root.val);
}

public List<Integer> rightSideView(TreeNode root){
    int n = lvl(root);
    List<Integer> ans = new ArrayList<>();
    for(int i=0; i<n; i++){
        ans.add(0);
    }
    dfs(root, 0, ans);
    return ans;
}
```

# LeetCode (110) { Balanced Binary Tree } #Most Optimized Code //



⇒ Code :-

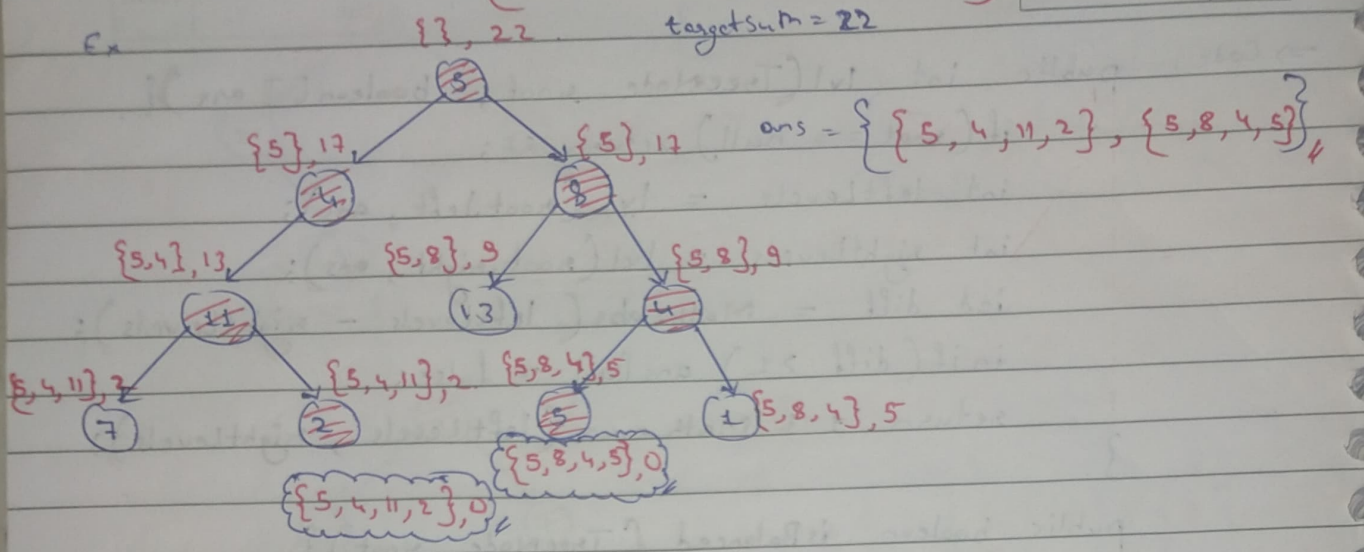
```
public int lvl(TreeNode root, boolean[] ans){
    if(root == null) return 0;
    int leftLevels = lvl(root.left, ans);
    int rightLevels = lvl(root.right, ans);
    ind diff = Math.abs(leftLevels - rightLevels);
    if(diff > 1) ans[0] = false;
    return 1 + Math.max(leftLevels, rightLevels);
}

public boolean isBalanced(TreeNode root){
    boolean[] ans = {true};
    lvl(root, ans);
    return ans[0];
}
```

# LeetCode (543) { Diameter of Binary Tree } #Most Optimized Code //

⇒ Code :-

```
public int levels(TreeNode root, int[] maxDia){
    if(root == null) return 0;
    int leftLevels = levels(root.left, maxDia);
    int rightLevels = levels(root.right, maxDia);
    ind dia = leftLevels + rightLevels;
    maxDia[0] = Math.max(dia, maxDia[0]);
    return 1 + Math.max(leftLevels, rightLevels);
}

public int diameterOfBinaryTree(TreeNode root){
    int[] maxDia = {0};
    levels(root, maxDia);
    return maxDia[0];
}
```

(11) LeetCode Q. No. (113) { Path Sum II }

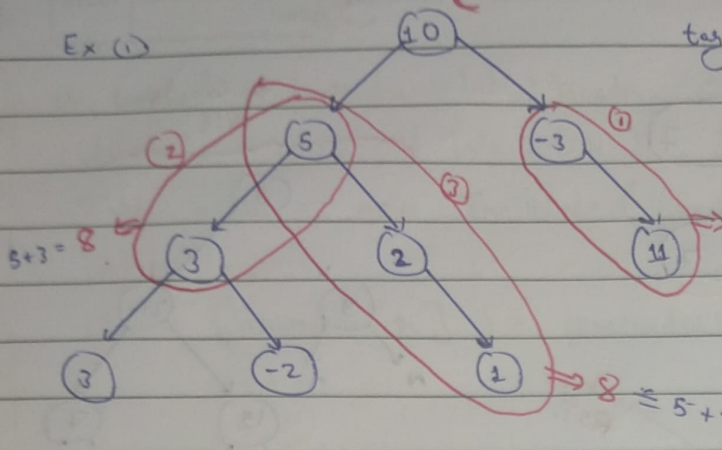



=> Code :-

```
public void helper(TreeNode root, int targetSum, List<Integer> arr, List<List<Integer>> ans){
    if(root == null) return;
    if(root.left == null && root.right == null && targetSum - root.val == 0){
        ans.add(arr);
    }
    arr.add(root.val);
    if(root.left == null && root.right == null && targetSum - root.val == 0){
        ans.add(arr);
    }
    helper(root.left, targetSum - root.val, new ArrayList<>(arr), ans);
    helper(root.right, targetSum - root.val, new ArrayList<>(arr), ans);
}

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

(12) LeetCode Q.No. (437) { Path Sum III }

Ex (1)

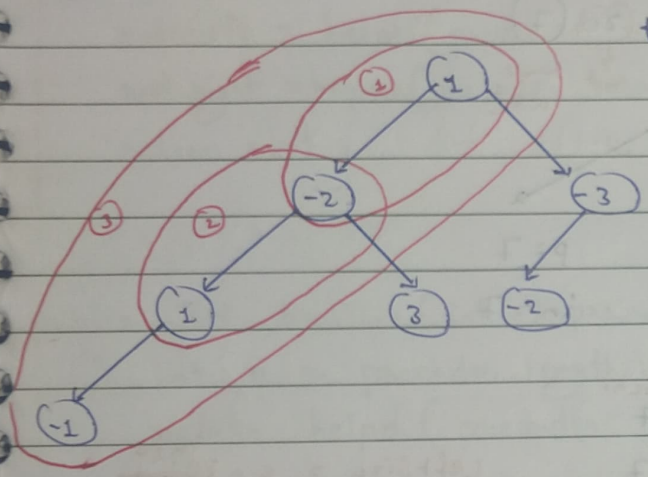



targetSum = (8)

Ans => (3)

Path does Not need to end at leaf Node

Ex (2)




targetSum = (-1)

Ans => (3)

=> Code :-

```
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 targetSum){
    if(root == null) return 0;
    return helper(root, targetSum) + pathSum(root.left, targetSum) + pathSum(root.right, targetSum);
}
```

(13) LeetCode Q. No. 105 { Construct Binary Tree from Preorder & Inorder Traversal }

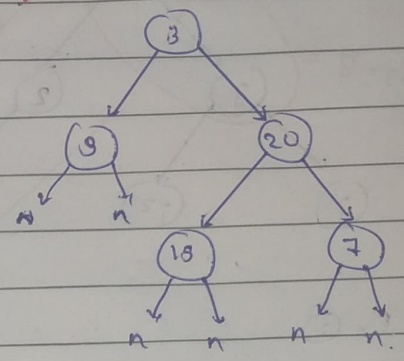
M T W T F S S | Page No.: | Date: | YOUVA

* Think About Root of the tree.

preorder = 3 | 9 | 20 15 7
root

inorder = 9 | 3 | 15 20 7
LST | root | RST

Need to Construct this.




root of LST

Pree = 9
in = 9.

root of RST.

Pre = 20 15 7
in = 15 | 20 | 7
LST | root | RST

pre = 15 | p = 7
in = 15 | in = 7

root Prelo Left ~~size~~ Prelo + ~~leftsize~~ Prelo + Leftsize+1 Prehi

pre = 3 | 9 | 20 | 15 | 7
in = 9 | 3 | 15 | 20 | 7
into | r. | inhi

LeftSize = r - into

~~LST~~

LST ⟶ prelo + 1 to prelo + leftsize , into to r - 1
RST ⟶ (prelo + leftsize + 1) to prehi , r + 1 to inhi.

---

LST ⟶ inlo to r - 1 , postlo to leftsize + postlo - 1.
RST ⟶ r + 1 to inhi , leftsize + postlo to posthi - 1

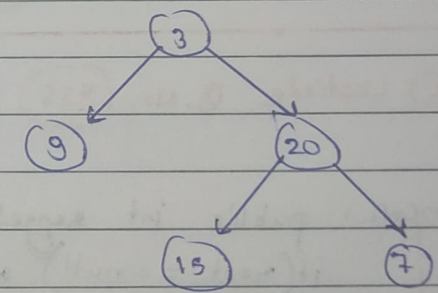
=> Code :

```
TN helper(int[] preorder, int[] inorder, int prelo, int prehi, int inlo, int inhi){
    if(prelo > prehi || inlo > inhi)  return null;
    TN root = new TreeNode(preorder[prelo]);
    int x = 0;
    while(inorder[x] != preorder[prelo]){
        x++;
    }
    int leftsize = x - inlo;
    root.left = helper(preorder, inorder, prelo+1, prelo+leftsize, inlo, x-1);
    root.right = helper(preorder, inorder, prelo+leftsize+1, prehi, x+1, inhi);
    return root;
}

public TN buildTree(int[] preorder, int[] inorder){
    int n = preorder.length;
    return helper(preorder, inorder, 0, n-1, 0, n-1);
}
```

(14) LeetCode Q.No. ~~150~~ 106 { Construct Binary Tree from Inorder & Postorder Traversal. } H.W.

inorder => 9 (3) 15 20 7 (root)
LST | RST

postorder => 9 15 7 20 (3) (root)
LST | RST




in => 9
post => (9) → root of LST

in => 15 (20) 7 (root)
LST | RST
post => 15 7 (20)
LST | RST. root of RST

in => (15)
post => (15) → root of LST

in => 7
post => (7) → Root of RST

Leftsize => x - inlo

| | inlo | x | | | inhi |
|---|---|---|---|---|---|
| in => | 9 | (3) | 15 | 20 | 7 |
| post => | 9 | 15 | 7 | 20 | (3) → root |
| | postlo | | | | posthi |

(Leftsize of Post: postlo to postlo-1) (leftsize + postlo)