**TREE ASSIGNMENT – 1 (20)**

Question-1

Given a binary tree, your task is to find a subtree with a maximum sum in the tree.

Examples:

Input1 :

1

/   \\

2      3

/ \    / \

4   5  6   7

Output1 : 28

As all the tree elements are positive, the largest subtree sum is equal to sum of all tree elements.

Input2 :

1

/    \\

-2      3

/ \    /  \

4   5  -6   2

Output2 : 7

Subtree with largest sum is :

-2

/ \

4   5

Also, entire tree sum is also 7.

Solution:

Class Node:

    def \_\_init\_\_(self, data):

        self.data = data

        #//assign data

        self.left = None

        self.right = None

    max=0

    ans = float("-infinity")

    #// float("infinity") is used for finding the lowest value

    def largestSubTreeSum(root):

        if root == None:

            return 0

        if root.left == None or root.right == None:

            return root.data

        leftsum = largestSubTreeSum(root.left)

        rightsum = largestSubTreeSum(root.right)

        rootnodesum = root.data + leftsum + rightsum

        tempmax = max(leftsum, rightsum)

        tempmax= max(tempmax, rootnodesum)

        #update the tempmax to ans

        global ans

        ans = max(ans, tempmax)

        return rootnodesum

Question-2

Construct the BST (Binary Search Tree) from its given level order traversal.

Example:

Input: arr[] = {7, 4, 12, 3, 6, 8, 1, 5, 10}

Output: BST:

7

/    \\

4     12

/  \\     /

3   6  8

/   /   \

1  5   10

Solution:

Class Node:

    def \_\_init\_\_(self, data):

        self.data = data

        self.left = None

        self.right = None

    #Create a NewNode

    def get\_newnode(data):

        new\_node = Node(data)

        new\_node.data = data

        new\_node.left = None

        new\_node.right =  None

        return new\_node

    # Create a function to make a BST with level Order Traversal ''

    def levelorderBST(root, data):

        if root == None:

            root = get\_newnode(data)

            return root

        if data < root.data:

            root.left = levelorderBST(root.left, data)

        else:

            root.right = levelorderBST(root.right, data)

        return root

    # Function to create Inorder Traversal

    def inOrderTraversal(root):

        if root == None:

            return None

        inOrderTraversal(root.left)

        print(root.data, end = " ")

        inOrderTraversal(root.right)

Question-3

Given an array of size n. The problem is to check whether the given array can represent the level order traversal of a Binary Search Tree or not.

Examples:

Input1 : arr[] = {7, 4, 12, 3, 6, 8, 1, 5, 10}

Output1 : Yes

For the given arr[], the Binary Search Tree is:

7

/    \\

4     12

/  \\     /

3   6  8

/   /   \

1  5   10

Input2 : arr[] = {11, 6, 13, 5, 12, 10}

Output2 : No

Approach :

We have used the queue data structure here;

We are storing the array elements in the queue one by one with their left and right child and then popping the element from the queue and checking if it’s smaller or greater than the arr next element and the min and max values we have pre-defined;

On the basis of that we are changing the node details and creating newNode. If it satisfies all the condition then it returns true else false.

INT\_MIN, INT\_MAX = float('-inf'), float('inf')

class nodeDetails:

    def \_\_init\_\_(self, data, min, max):

        self.data = data

        self.min = min

        self.max = max

    def BinaryTreelevelOrder(arr, n):

        if n == 0:

            return True

        q = []

        i = 0

        newNode = nodeDetails(arr[i], INT\_MIN, INT\_MAX)

        i +=1

        q.append(newNode)

        while i<n and len(q):

            temp = q.pop(0)

            if i<n and (arr[i] < temp.data and arr[i]> temp.min) :

                newNode = nodeDetails(arr[i], temp.min, temp.data)

                i += 1

                q.append(newNode)

            if i<n and (arr[i] > temp.data and arr[i] < temp.max):

                newNode = nodeDetails(arr[i], temp.data, temp.max)

                i += 1

                q.append(newNode)

            if i == n :

                return True

            return False