**Binary Tree to BST: -**

Easy Accuracy: 60.75% Submissions: 57K+ Points: 2

Given a Binary Tree, convert it to Binary Search Tree in such a way **that keeps the original structure of Binary Tree intact**.

**Example 1:**

**Input:**

1

  / \

2 3

**Output:**1 2 3  
**Explanation:**  
The converted BST will be   
 2  
 / \  
 1 3

**Example 2:**

**Input:**

1

/ \

2 3

/

4

**Output:**1 2 3 4

**Explanation:**

The converted BST will be

3

/ \

2 4

/

1

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function **binaryTreeToBST()** which takes the root of the Binary tree as input and returns the root of the BST. The driver code will print**inorder** traversal of the converted BST.

**Expected Time Complexity:** O(NLogN).  
**Expected Auxiliary Space:** O(N).

**Constraints:**  
1 <= Number of nodes <= 105

**Code: -**

//{ Driver Code Starts

//Initial template for C++

#include <bits/stdc++.h>

using namespace std;

struct Node

{

int data;

struct Node \*left;

struct Node \*right;

Node(int val) {

data = val;

left = right = NULL;

}

};

void printInorder (struct Node\* node)

{

if (node == NULL)

return;

printInorder (node->left);

printf("%d ", node->data);

printInorder (node->right);

}

// 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 = new Node(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 = new Node(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 = new Node(stoi(currVal));

// Push it to the queue

queue.push(currNode->right);

}

i++;

}

return root;

}

// } Driver Code Ends

//User function template for C++

/\* The Node structure is

struct Node

{

int data;

struct Node \*left;

struct Node \*right;

Node(int val) {

data = val;

left = right = NULL;

}

};

\*/

class Solution{

public:

void insert(Node \*root, vector<int> &node){

if(!root)

return;

insert(root->left, node);

node.push\_back(root->data);

insert(root->right, node);

return;

}

void make(Node \*root, vector<int> &node, int &ind){

if(!root)

return;

make(root->left, node, ind);

root->data = node[ind++];

make(root->right, node, ind);

return;

}

// The given root is the root of the Binary Tree

// Return the root of the generated BST

Node \*binaryTreeToBST (Node \*root){

if(!root->left and !root->right)

return root;

vector<int> node;

insert(root, node);

sort(node.begin(), node.end());

int ind = 0;

make(root, node, ind);

return root;

}

};

//{ Driver Code Starts.

int main()

{

int t;

scanf("%d\n", &t);

while (t--)

{

string s;

getline(cin,s);

Node\* root = buildTree(s);

Solution obj;

Node \*res = obj.binaryTreeToBST (root);

printInorder(res);

cout<<endl;

}

return 0;

}

// } Driver Code Ends

**T.C: - O(N \* log N)**

**S.C: - O(N)**