**Insert a node in a BST: -**

Easy Accuracy: 47.15% Submissions: 115K+ Points: 2

Given a **BST** and a key **K**. If K is not present in the BST, Insert a new Node with a value equal to K into the BST. If K is already present in the BST, don't modify the BST.

**Example 1:**

**Input:**

     2

  /   \   1    3

K = 4

**Output:**1 2 3 4

**Explanation:**After inserting the node 4

Inorder traversal will be 1 2 3 4.

**Example 2:**

**Input:**

        2

      /   \

    1     3

         \

          6

K = 4

**Output:**1 2 3 4 6

**Explanation:**After inserting the node 4

Inorder traversal of the above tree will be 1 2 3 4 6.

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function **insert()** which takes the root of the BST and Key K as input parameters and returns the root of the modified BST after inserting K.   
**Note:**The generated output contains the inorder traversal of the modified tree.

**Expected Time Complexity:** O(Height of the BST).  
**Expected Auxiliary Space:** O(Height of the BST).

**Constraints:**  
1 <= Number of nodes initially in BST <= 105  
1 <= K <= 109

**Code: -**

//{ Driver Code Starts

#include <bits/stdc++.h>

using namespace std;

#define MAX\_HEIGHT 100000

// Tree Node

struct Node {

int data;

Node\* left;

Node\* right;

Node(int val) {

data = val;

left = right = NULL;

}

};

// } Driver Code Ends

// Function to insert a node in a BST.

/\*

struct Node {

int data;

Node\* left;

Node\* right;

Node(int val) {

data = val;

left = right = NULL;

}

};

\*/

class Solution

{

public:

Node\* insert(Node\* node, int data){

if(!node){

Node \*root = new Node(data);

return root;

}

else if(data < node->data)

node->left = insert(node->left, data);

else if(data > node->data)

node->right = insert(node->right, data);

// I don't care for the remaining case like:-

// data == node->data

return node;

}

};

//{ Driver Code Starts.

// 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;

}

void inorder(Node\* root, vector<int>& v) {

if (root == NULL) return;

inorder(root->left, v);

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

inorder(root->right, v);

}

int main() {

int t;

string tc;

getline(cin, tc);

t = stoi(tc);

while (t--) {

string s;

getline(cin, s);

Node\* root = buildTree(s);

getline(cin, s);

int k = stoi(s);

// getline(cin, s);

Solution ob;

ob.insert(root, k);

vector<int> v;

inorder(root, v);

for (int i = 0; i < v.size(); i++) cout << v[i] << " ";

cout << endl;

// cout<<"~"<<endl;

}

return 0;

}

// } Driver Code Ends

**T.C: - O(height of BST)**

**S.C: - O(height of BST)**