**Kth largest element in BST: -**

Easy Accuracy: 49.31% Submissions: 111K+ Points: 2

Given a **Binary Search Tree**. Your task is to complete the function which will return the **Kth largest** element without doing any modification in Binary Search Tree.

**Example 1:**

**Input:**

**4**

  / \

2 9

k = 2

**Output:** 4

**Example 2:**

**Input:**

       9

       \

**10**

K = 1

**Output:** 10

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function **kthLargest()** which takes the root of the BST and an integer K as inputs and returns the Kth largest element in the given BST.

**Expected Time Complexity:** O(N).  
**Expected Auxiliary Space:** O(H) where H is max recursion stack of height H at a given time.

**Constraints:**  
1 <= N <= 105  
1 <= K <= N

**Code: -**

//{ Driver Code Starts

#include <bits/stdc++.h>

using namespace std;

// Tree Node

struct Node {

int data;

Node \*left;

Node \*right;

Node(int val) {

data = val;

left = right = NULL;

}

};

// 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

/\*The Node structure is defined as

struct Node {

int data;

Node \*left;

Node \*right;

Node(int val) {

data = val;

left = right = NULL;

}

};

\*/

// return the Kth largest element in the given BST rooted at 'root'

class Solution

{

public:

int kthLargest(Node \*root, int &K){

if(!root) return INT\_MAX;

int ansright = kthLargest(root->right, K);

if(ansright != INT\_MAX)

return ansright;

--K;

if(K==0) return root->data;

int ansleft = kthLargest(root->left, K);

return ansleft;

}

};

//{ Driver Code Starts.

int main()

{

int t;

cin>>t;

getchar();

while(t--)

{

string s;

getline(cin,s);

Node\* head = buildTree(s);

int k;

cin>>k;

getchar();

Solution ob;

cout << ob.kthLargest( head, k ) << endl;

}

return 1;

}

// } Driver Code Ends

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

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