**Leaf under budget: -**

Easy Accuracy: 50.05% Submissions: 22K+ Points: 2

Given a binary tree and a **budget**. Assume you are at the root of the tree**(level 1)**, you need to maximise the count of leaf nodes you can visit in your budget if the **cost of visiting**a leaf node is equal to the **level of that leaf node**.

**Example 1:**

**Input:**

10

/ \

8 2

/ / \

3 3 6

\

4

and budget = 8

**Output: 2**

**Explanation:**

Cost For visiting Leaf Node 3: 3

Cost For visiting Leaf Node 4: 4

Cost For visiting Leaf Node 6: 3

In budget 8 one can visit Max 2 Leaf Nodes.

**Example 2:**

**Input:**

1

  / \

  2 3

  / \ / \

  4 5 6 7

and budget = 5

**Output:** 1  
**Explanation:** We can only visit either node 4 or 5.

**Your Task:**

You don't need to read input or print anything. Your task is to complete the function **getCount()**which takes root node of the tree and a integer denoting the budget as input parameters and returns an integer denoting the count of visited leaf nodes of the tree.

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

**Constraints:**  
1<=N<=105  
1<=budget<=104

**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 x)

{

data = x;

left = NULL;

right = NULL;

}

};

void printInorder(Node \*node)

{

if (node == NULL)

{

return;

}

printInorder(node->left);

cout << node->data << " ";

printInorder(node->right);

}

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

/\*

struct Node

{

int data;

struct Node \*left;

struct Node \*right;

Node(int x)

{

data = x;

left = NULL;

right = NULL;

}

};

\*/

class Solution

{

public:

void dfs(Node \*root, int level, vector<int> &temp){

if(root->left)

dfs(root->left, level+1, temp);

if(root->right)

dfs(root->right, level+1, temp);

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

temp.push\_back(level);

return;

}

int getCount(Node \*root, int k){

vector<int> temp;

dfs(root, 1, temp);

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

int count=0;

for(int i=0; i<temp.size(); ++i){

if(k-temp[i] >= 0)

++count;

else

break;

k -= temp[i];

}

return count;

}

};

//{ Driver Code Starts.

int main()

{

int t;

scanf("%d", &t);

cin.ignore();

while (t--)

{

string treeString;

getline(cin, treeString);

Node \*root = buildTree(treeString);

int k;

cin >> k;

cin.ignore();

Solution obj;

int res = obj.getCount(root, k);

cout << res << "\n";

}

return 0;

}

// } Driver Code Ends

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

**S.C: - O(k\*log k); k = no. of leaf**