**Maximum Depth Of Binary Tree :-**

Easy Accuracy: 79.53% Submissions: 12K+ Points: 2

Given a binary tree, find its **maximum** **depth**.  
A binary tree's maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

**Example 1:**

**Input:**  
 root  -->     1  
                      /   \  
                    3      2  
                  /  
                4             
**Output:** 3  
**Explanation:**  
Maximum depth is between nodes 1 and 4, which is 3.

**Example 2:**

**Input:**

root --> 10

/ \

20 30

\ \

40 60

/

2

**Output:** 4

**Explanation:**

Maximum depth is between nodes 10 and 2, which is 4

**Your Task:**  
You don't need to read input or print anything. Complete the function **maxDepth()** which takes the root node as an input parameter and returns the maximum depth.  
   
**Expected Time Complexity:**O(N)  
**Expected Auxiliary Space:**O(height of the tree)  
   
**Constraints:**  
1 ≤ N ≤ 10^5

**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;

};

// Utility function to create a new Tree Node

Node\* newNode(int val) {

Node\* temp = new Node;

temp->data = val;

temp->left = NULL;

temp->right = NULL;

return temp;

}

// 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 = newNode(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 = newNode(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 = newNode(stoi(currVal));

// Push it to the queue

queue.push(currNode->right);

}

i++;

}

return root;

}

// } Driver Code Ends

class Solution{

public:

/\*You are required to complete this method\*/

int maxDepth(Node \*root) {

queue<Node \*> q;

Node \*ptr=NULL;

int ans=1;

if(root->left)

q.push(root->left);

if(root->right)

q.push(root->right);

while(q.empty()==false){

int count = q.size();

ans++;

while(count--){

ptr = q.front();

q.pop();

if(ptr->left)

q.push(ptr->left);

if(ptr->right)

q.push(ptr->right);

}

}

return ans;

}

};

//{ Driver Code Starts.

int main() {

int t;

string tc;

getline(cin, tc);

t = stoi(tc);

while (t--) {

string s, ch;

getline(cin, s);

Node\* root = buildTree(s);

// getline(cin, ch);

Solution obj;

cout << obj.maxDepth(root) << endl;

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

}

return 0;

}

// } Driver Code Ends

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

**S.C :- O(max no. of nodes present in any level)**