**Leftmost and rightmost nodes of binary tree: -**

Medium Accuracy: 52.58% Submissions: 26K+ Points: 4

Given a Binary Tree of size N, Print the corner nodes ie- the node at the leftmost and rightmost of each level.

**Example 1:**

**Input :**

1

/ \

2 3

/ \ / \

4 5 6 7

**Output:** 1 2 3 4 7

**Explanation:**

Corners at level 0: 1

Corners at level 1: 2 3

Corners at level 2: 4 7

**Example 2:**

**Input:**

10

/ \

20 30

/ \

40 60

**Output:** 10 20 30 40 60

**Your Task:**  
You dont need to read input. Complete the function **printCorner()**which takes root node as input parameter and prints the corner nodes separated by spaces. The left corner should be printed before the right for each level starting from level 0.  
**Note:**Don't print a new line after printing all the corner nodes.

**Expected Time Complexity:** O(N)  
**Expected Auxiliary Space:** O(number of nodes in a level)

**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;

}

void printCorner(Node \*root);

// 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;

}

int main() {

int t;

string tc;

getline(cin, tc);

t=stoi(tc);

while(t--)

{

string s ,ch;

getline(cin, s);

Node\* root = buildTree(s);

printCorner(root);

cout<<endl;

}

return 0;

}

// } Driver Code Ends

/\* Function to print corner node at each level \*/

/\*

Structure of a node is as following

struct Node

{

int data;

struct Node\* left;

struct Node\* right;

};

\*/

void printCorner(Node \*root){

queue<Node \*> q;

q.push(root);

while(q.size()){

int count = q.size();

for(int i=1; i<=count; ++i){

Node \*front = q.front();

q.pop();

if(i==1 or i==count)

cout << front->data << " ";

if(front->left)

q.push(front->left);

if(front->right)

q.push(front->right);

}

}

}

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

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