**Largest value in each level of Binary Tree | Set-2 (Iterative Approach)**

**C:**

// C++ implementation to print largest

// value in each level of Binary Tree

#include <bits/stdc++.h>

using namespace std;

// structure of a node of binary tree

struct Node {

int data;

Node \*left, \*right;

};

// function to get a new node

Node\* newNode(int data)

{

// allocate space

Node\* temp = new Node;

// put in the data

temp->data = data;

temp->left = temp->right = NULL;

return temp;

}

// function to print largest value

// in each level of Binary Tree

void largestValueInEachLevel(Node\* root)

{

// if tree is empty

if (!root)

return;

queue<Node\*> q;

int nc, max;

// push root to the queue 'q'

q.push(root);

while (1) {

// node count for the current level

nc = q.size();

// if true then all the nodes of

// the tree have been traversed

if (nc == 0)

break;

// maximum element for the current

// level

max = INT\_MIN;

while (nc--) {

// get the front element from 'q'

Node\* front = q.front();

// remove front element from 'q'

q.pop();

// if true, then update 'max'

if (max < front->data)

max = front->data;

// if left child exists

if (front->left)

q.push(front->left);

// if right child exists

if (front->right)

q.push(front->right);

}

// print maximum element of

// current level

cout << max << " ";

}

}

// Driver code

int main()

{

/\* Construct a Binary Tree

4

/ \

9 2

/ \ \

3 5 7 \*/

Node\* root = NULL;

root = newNode(4);

root->left = newNode(9);

root->right = newNode(2);

root->left->left = newNode(3);

root->left->right = newNode(5);

root->right->right = newNode(7);

// Function call

largestValueInEachLevel(root);

return 0;

}

**Java:**

// Java implementation to print largest

// value in each level of Binary Tree

import java.util.\*;

class GfG {

// structure of a node of binary tree

static class Node

{

int data;

Node left = null;

Node right = null;

}

// function to get a new node

static Node newNode(int val)

{

// allocate space

Node temp = new Node();

// put in the data

temp.data = val;

temp.left = null;

temp.right = null;

return temp;

}

// function to print largest value

// in each level of Binary Tree

static void largestValueInEachLevel(Node root)

{

// if tree is empty

if (root == null)

return;

Queue<Node> q = new LinkedList<Node>();

int nc, max;

// push root to the queue 'q'

q.add(root);

while (true)

{

// node count for the current level

nc = q.size();

// if true then all the nodes of

// the tree have been traversed

if (nc == 0)

break;

// maximum element for the current

// level

max = Integer.MIN\_VALUE;

while (nc != 0)

{

// get the front element from 'q'

Node front = q.peek();

// remove front element from 'q'

q.remove();

// if true, then update 'max'

if (max < front.data)

max = front.data;

// if left child exists

if (front.left != null)

q.add(front.left);

// if right child exists

if (front.right != null)

q.add(front.right);

nc--;

}

// print maximum element of

// current level

System.out.println(max + " ");

}

}

// Driver code

public static void main(String[] args)

{

/\* Construct a Binary Tree

4

/ \

9 2

/ \ \

3 5 7 \*/

Node root = null;

root = newNode(4);

root.left = newNode(9);

root.right = newNode(2);

root.left.left = newNode(3);

root.left.right = newNode(5);

root.right.right = newNode(7);

// Function call

largestValueInEachLevel(root);

}

}

Js:

<script>

// JavaScript implementation to print largest

// value in each level of Binary Tree

// structure of a node of binary tree

class Node

{

constructor(data) {

this.left = null;

this.right = null;

this.data = data;

}

}

// function to get a new node

function newNode(val)

{

// allocate space

let temp = new Node(val);

return temp;

}

// function to print largest value

// in each level of Binary Tree

function largestValueInEachLevel(root)

{

// if tree is empty

if (root == null)

return;

let q = [];

let nc, max;

// push root to the queue 'q'

q.push(root);

while (true)

{

// node count for the current level

nc = q.length;

// if true then all the nodes of

// the tree have been traversed

if (nc == 0)

break;

// maximum element for the current

// level

max = Number.MIN\_VALUE;

while (nc != 0)

{

// get the front element from 'q'

let front = q[0];

// remove front element from 'q'

q.shift();

// if true, then update 'max'

if (max < front.data)

max = front.data;

// if left child exists

if (front.left != null)

q.push(front.left);

// if right child exists

if (front.right != null)

q.push(front.right);

nc--;

}

// print maximum element of

// current level

document.write(max + " ");

}

}

/\* Construct a Binary Tree

4

/ \

9 2

/ \ \

3 5 7 \*/

let root = null;

root = newNode(4);

root.left = newNode(9);

root.right = newNode(2);

root.left.left = newNode(3);

root.left.right = newNode(5);

root.right.right = newNode(7);

// Function call

largestValueInEachLevel(root);

</script>