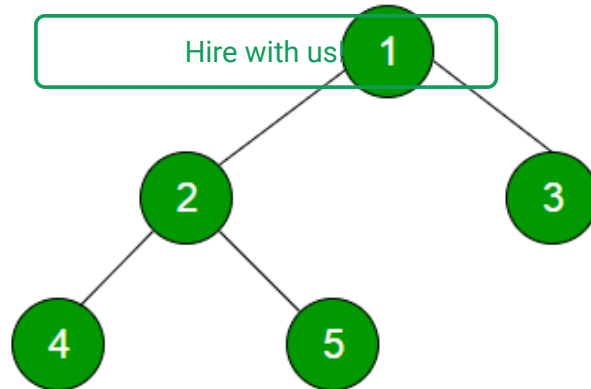


Level Order Tree Traversal

Level order traversal of a tree :

Custom Search

Hire with us



Level order traversal of the above tree is 1 2 3 4 5



Get the :
you de
in

Level up yo
with
Science I

Learn



Recommended: Please solve it on “PRACTICE” first, before moving on to the solution.

Method 1 (Use function to print a given level)

Algorithm:

There are basically two functions in this method. One is to print all nodes at a given level (printGivenLevel), and other is to print level order traversal of the tree (printLevelorder). printLevelorder makes use of printGivenLevel to print nodes at all levels one by one starting from root.

```

/*Function to print level order traversal of tree*/
printLevelorder(tree)
for d = 1 to height(tree)
    printGivenLevel(tree, d);

/*Function to print all nodes at a given level*/
printGivenLevel(tree, level)
if tree is NULL then return;
if level is 1, then
    print(tree->data);
else if level greater than 1, then
    printGivenLevel(tree->left, level-1);
    printGivenLevel(tree->right, level-1);

```

Implementation:

C++

```

// Recursive CPP program for level
// order traversal of Binary Tree
#include <bits/stdc++.h>
using namespace std;

/* A binary tree node has data,
pointer to left child
and a pointer to right child */
class node
{
public:
    int data;
    node* left, *right;
};

/* Function prototypes */
void printGivenLevel(node* root, int level);
int height(node* node);
node* newNode(int data);

/* Function to print level
order traversal a tree*/
void printLevelOrder(node* root)
{
    int h = height(root);
    int i;
    for (i = 1; i <= h; i++)
        printGivenLevel(root, i);
}

/* Print nodes at a given level */
void printGivenLevel(node* root, int level)
{
    if (root == NULL)
        return;

```

```

    if (level == 1)
        cout << root->data << " ";
    else if (level > 1)
    {
        printGivenLevel(root->left, level-1);
        printGivenLevel(root->right, level-1);
    }
}

```

```

/* Compute the "height" of a tree -- the number of
   nodes along the longest path from the root node
   down to the farthest leaf node.*/

```

```

int height(node* node)
{
    if (node == NULL)
        return 0;
    else
    {
        /* compute the height of each subtree */
        int lheight = height(node->left);
        int rheight = height(node->right);

        /* use the larger one */
        if (lheight > rheight)
            return(lheight + 1);
        else return(rheight + 1);
    }
}

```

```

/* Helper function that allocates
   a new node with the given data and
   NULL left and right pointers. */

```

```

node* newNode(int data)
{
    node* Node = new node();
    Node->data = data;
    Node->left = NULL;
    Node->right = NULL;

    return(Node);
}

```

```

/* Driver code*/

```

```

int main()
{
    node *root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);

    cout << "Level Order traversal of binary tree is \n";
    printLevelOrder(root);

    return 0;
}

```

```

// This code is contributed by rathbhupendra

```

C

```
// Recursive C program for level order traversal of Binary Tree
#include <stdio.h>
#include <stdlib.h>

/* A binary tree node has data, pointer to left child
   and a pointer to right child */
struct node
{
    int data;
    struct node* left, *right;
};

/* Function prototypes */
void printGivenLevel(struct node* root, int level);
int height(struct node* node);
struct node* newNode(int data);

/* Function to print level order traversal a tree*/
void printLevelOrder(struct node* root)
{
    int h = height(root);
    int i;
    for (i=1; i<=h; i++)
        printGivenLevel(root, i);
}

/* Print nodes at a given level */
void printGivenLevel(struct node* root, int level)
{
    if (root == NULL)
        return;
    if (level == 1)
        printf("%d ", root->data);
    else if (level > 1)
    {
        printGivenLevel(root->left, level-1);
        printGivenLevel(root->right, level-1);
    }
}

/* Compute the "height" of a tree -- the number of
   nodes along the longest path from the root node
   down to the farthest leaf node.*/
int height(struct node* node)
{
    if (node==NULL)
        return 0;
    else
    {
        /* compute the height of each subtree */
        int lheight = height(node->left);
        int rheight = height(node->right);

        /* use the larger one */
        if (lheight > rheight)
```

```

        return(lheight+1);
    else return(rheight+1);
}
}

/* Helper function that allocates a new node with the
given data and NULL left and right pointers. */
struct node* newNode(int data)
{
    struct node* node = (struct node*)
                        malloc(sizeof(struct node));
    node->data = data;
    node->left = NULL;
    node->right = NULL;

    return(node);
}

/* Driver program to test above functions*/
int main()
{
    struct node *root = newNode(1);
    root->left      = newNode(2);
    root->right     = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);

    printf("Level Order traversal of binary tree is \n");
    printLevelOrder(root);

    return 0;
}

```

→

Java

```

// Recursive Java program for level order traversal of Binary Tree

/* Class containing left and right child of current
node and key value*/
class Node
{
    int data;
    Node left, right;
    public Node(int item)
    {
        data = item;
        left = right = null;
    }
}

class BinaryTree
{
    // Root of the Binary Tree
    Node root;

    public BinaryTree()
    {

```

```

        root = null;
    }

    /* function to print level order traversal of tree*/
    void printLevelOrder()
    {
        int h = height(root);
        int i;
        for (i=1; i<=h; i++)
            printGivenLevel(root, i);
    }

    /* Compute the "height" of a tree -- the number of
    nodes along the longest path from the root node
    down to the farthest leaf node.*/
    int height(Node root)
    {
        if (root == null)
            return 0;
        else
        {
            /* compute height of each subtree */
            int lheight = height(root.left);
            int rheight = height(root.right);

            /* use the larger one */
            if (lheight > rheight)
                return(lheight+1);
            else return(rheight+1);
        }
    }

    /* Print nodes at the given level */
    void printGivenLevel (Node root ,int level)
    {
        if (root == null)
            return;
        if (level == 1)
            System.out.print(root.data + " ");
        else if (level > 1)
        {
            printGivenLevel(root.left, level-1);
            printGivenLevel(root.right, level-1);
        }
    }

    /* Driver program to test above functions */
    public static void main(String args[])
    {
        BinaryTree tree = new BinaryTree();
        tree.root= new Node(1);
        tree.root.left= new Node(2);
        tree.root.right= new Node(3);
        tree.root.left.left= new Node(4);
        tree.root.left.right= new Node(5);

        System.out.println("Level order traversal of binary tree is ");
        tree.printLevelOrder();
    }

```

```
}
```

Python

```
# Recursive Python program for level order traversal of Binary Tree
```

```
# A node structure
```

```
class Node:
```

```
    # A utility function to create a new node
```

```
    def __init__(self, key):
```

```
        self.data = key
```

```
        self.left = None
```

```
        self.right = None
```

```
# Function to print level order traversal of tree
```

```
def printLevelOrder(root):
```

```
    h = height(root)
```

```
    for i in range(1, h+1):
```

```
        printGivenLevel(root, i)
```

```
# Print nodes at a given level
```

```
def printGivenLevel(root , level):
```

```
    if root is None:
```

```
        return
```

```
    if level == 1:
```

```
        print "%d" %(root.data),
```

```
    elif level > 1 :
```

```
        printGivenLevel(root.left , level-1)
```

```
        printGivenLevel(root.right , level-1)
```

```
""" Compute the height of a tree--the number of nodes
    along the longest path from the root node down to
    the farthest leaf node
    """
```

```
def height(node):
```

```
    if node is None:
```

```
        return 0
```

```
    else :
```

```
        # Compute the height of each subtree
```

```
        lheight = height(node.left)
```

```
        rheight = height(node.right)
```

```
        #Use the larger one
```

```
        if lheight > rheight :
```

```
            return lheight+1
```

```
        else:
```

```
            return rheight+1
```

```
# Driver program to test above function
```

```
root = Node(1)
```

```
root.left = Node(2)
```

```
root.right = Node(3)
```

```
root.left.left = Node(4)
```

```
root.left.right = Node(5)
```

```
print "Level order traversal of binary tree is -"  
printLevelOrder(root)
```

```
#This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

C#

```
// Recursive c# program for level  
// order traversal of Binary Tree  
using System;  
  
/* Class containing left and right  
   child of current node and key value*/  
public class Node  
{  
    public int data;  
    public Node left, right;  
    public Node(int item)  
    {  
        data = item;  
        left = right = null;  
    }  
}  
  
class GFG  
{  
    // Root of the Binary Tree  
    public Node root;  
  
    public void BinaryTree()  
    {  
        root = null;  
    }  
  
    /* function to print level order  
       traversal of tree*/  
    public virtual void printLevelOrder()  
    {  
        int h = height(root);  
        int i;  
        for (i = 1; i <= h; i++)  
        {  
            printGivenLevel(root, i);  
        }  
    }  
  
    /* Compute the "height" of a tree --  
       the number of nodes along the longest  
       path from the root node down to the  
       farthest leaf node.*/  
    public virtual int height(Node root)  
    {  
        if (root == null)  
        {  
            return 0;  
        }  
    }  
}
```



```

    }
    else
    {
        /* compute height of each subtree */
        int lheight = height(root.left);
        int rheight = height(root.right);

        /* use the larger one */
        if (lheight > rheight)
        {
            return (lheight + 1);
        }
        else
        {
            return (rheight + 1);
        }
    }
}

/* Print nodes at the given level */
public virtual void printGivenLevel(Node root,
                                     int level)
{
    if (root == null)
    {
        return;
    }
    if (level == 1)
    {
        Console.Write(root.data + " ");
    }
    else if (level > 1)
    {
        printGivenLevel(root.left, level - 1);
        printGivenLevel(root.right, level - 1);
    }
}

```

// Driver Code

```

public static void Main(string[] args)
{
    GFG tree = new GFG();
    tree.root = new Node(1);
    tree.root.left = new Node(2);
    tree.root.right = new Node(3);
    tree.root.left.left = new Node(4);
    tree.root.left.right = new Node(5);

    Console.WriteLine("Level order traversal " +
                     "of binary tree is ");
    tree.printLevelOrder();
}
}

```

// This code is contributed by Shrikant13

Output:

Level order traversal of binary tree is -
1 2 3 4 5

Time Complexity: $O(n^2)$ in worst case. For a skewed tree, printGivenLevel() takes $O(n)$ time where n is the number of nodes in the skewed tree. So time complexity of printLevelOrder() is $O(n) + O(n-1) + O(n-2) + \dots + O(1)$ which is $O(n^2)$.

Space Complexity: $O(n)$ in worst case. For a skewed tree, printGivenLevel() uses $O(n)$ space for call stack. For a Balanced tree, call stack uses $O(\log n)$ space, (i.e., height of the balanced tree).

Method 2 (Using queue)

Algorithm:

For each node, first the node is visited and then its child nodes are put in a FIFO queue.

```
printLevelorder(tree)
1) Create an empty queue q
2) temp_node = root /*start from root*/
3) Loop while temp_node is not NULL
    a) print temp_node->data.
    b) Enqueue temp_node's children (first left then right children) to q
    c) Dequeue a node from q and assign its value to temp_node
```

Implementation:

Here is a simple implementation of the above algorithm. Queue is implemented using an array with maximum size of 500. We can implement queue as linked list also.

C++

```
/* C++ program to print level order traversal using STL */
#include <iostream>
```

```

#include <queue>
using namespace std;

// A Binary Tree Node
struct Node
{
    int data;
    struct Node *left, *right;
};

// Iterative method to find height of Binary Tree
void printLevelOrder(Node *root)
{
    // Base Case
    if (root == NULL) return;

    // Create an empty queue for level order traversal
    queue<Node *> q;

    // Enqueue Root and initialize height
    q.push(root);

    while (q.empty() == false)
    {
        // Print front of queue and remove it from queue
        Node *node = q.front();
        cout << node->data << " ";
        q.pop();

        /* Enqueue left child */
        if (node->left != NULL)
            q.push(node->left);

        /* Enqueue right child */
        if (node->right != NULL)
            q.push(node->right);
    }
}

// Utility function to create a new tree node
Node* newNode(int data)
{
    Node *temp = new Node;
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}

// Driver program to test above functions
int main()
{
    // Let us create binary tree shown in above diagram
    Node *root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);

    cout << "Level Order traversal of binary tree is \n";
}

```

```

    printLevelOrder(root);
    return 0;
}

```

C

```

// Iterative Queue based C program to do level order traversal
// of Binary Tree
#include <stdio.h>
#include <stdlib.h>
#define MAX_Q_SIZE 500

/* A binary tree node has data, pointer to left child
   and a pointer to right child */
struct node
{
    int data;
    struct node* left;
    struct node* right;
};

/* function prototypes */
struct node** createQueue(int *, int *);
void enqueue(struct node **, int *, struct node *);
struct node *deQueue(struct node **, int *);

/* Given a binary tree, print its nodes in level order
   using array for implementing queue */
void printLevelOrder(struct node* root)
{
    int rear, front;
    struct node **queue = createQueue(&front, &rear);
    struct node *temp_node = root;

    while (temp_node)
    {
        printf("%d ", temp_node->data);

        /*Enqueue left child */
        if (temp_node->left)
            enqueue(queue, &rear, temp_node->left);

        /*Enqueue right child */
        if (temp_node->right)
            enqueue(queue, &rear, temp_node->right);

        /*Dequeue node and make it temp_node*/
        temp_node = deQueue(queue, &front);
    }
}

/*UTILITY FUNCTIONS*/
struct node** createQueue(int *front, int *rear)
{
    struct node **queue =
        (struct node **)malloc(sizeof(struct node*)*MAX_Q_SIZE);
}

```

```

    *front = *rear = 0;
    return queue;
}

void enqueue(struct node **queue, int *rear, struct node *new_node)
{
    queue[*rear] = new_node;
    (*rear)++;
}

struct node *deQueue(struct node **queue, int *front)
{
    (*front)++;
    return queue[*front - 1];
}

/* Helper function that allocates a new node with the
   given data and NULL left and right pointers. */
struct node* newNode(int data)
{
    struct node* node = (struct node*)
                        malloc(sizeof(struct node));
    node->data = data;
    node->left = NULL;
    node->right = NULL;

    return(node);
}

/* Driver program to test above functions*/
int main()
{
    struct node *root = newNode(1);
    root->left      = newNode(2);
    root->right     = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);

    printf("Level Order traversal of binary tree is \n");
    printLevelOrder(root);

    return 0;
}

```

Java

```

// Iterative Queue based Java program to do level order traversal
// of Binary Tree

/* importing the inbuilt java classes required for the program */
import java.util.Queue;
import java.util.LinkedList;

/* Class to represent Tree node */
class Node {
    int data;
    Node left, right;
}

```

```

    public Node(int item) {
        data = item;
        left = null;
        right = null;
    }
}

/* Class to print Level Order Traversal */
class BinaryTree {

    Node root;

    /* Given a binary tree. Print its nodes in level order
    using array for implementing queue */
    void printLevelOrder()
    {
        Queue<Node> queue = new LinkedList<Node>();
        queue.add(root);
        while (!queue.isEmpty())
        {

            /* poll() removes the present head.
            For more information on poll() visit
            http://www.tutorialspoint.com/java/util/linkedlist_poll.htm */
            Node tempNode = queue.poll();
            System.out.print(tempNode.data + " ");

            /*Enqueue left child */
            if (tempNode.left != null) {
                queue.add(tempNode.left);
            }

            /*Enqueue right child */
            if (tempNode.right != null) {
                queue.add(tempNode.right);
            }
        }
    }

    public static void main(String args[])
    {
        /* creating a binary tree and entering
        the nodes */
        BinaryTree tree_level = new BinaryTree();
        tree_level.root = new Node(1);
        tree_level.root.left = new Node(2);
        tree_level.root.right = new Node(3);
        tree_level.root.left.left = new Node(4);
        tree_level.root.left.right = new Node(5);

        System.out.println("Level order traversal of binary tree is - ");
        tree_level.printLevelOrder();
    }
}

```

```
# Python program to print level order traversal using Queue
```

```
# A node structure
```

```
class Node:
```

```
    # A utility function to create a new node
```

```
    def __init__(self ,key):
```

```
        self.data = key
```

```
        self.left = None
```

```
        self.right = None
```

```
# Iterative Method to print the height of binary tree
```

```
def printLevelOrder(root):
```

```
    # Base Case
```

```
    if root is None:
```

```
        return
```

```
    # Create an empty queue for level order traversal
```

```
    queue = []
```

```
    # Enqueue Root and initialize height
```

```
    queue.append(root)
```

```
while(len(queue) > 0):
```

```
    # Print front of queue and remove it from queue
```

```
    print queue[0].data,
```

```
    node = queue.pop(0)
```

```
    #Enqueue left child
```

```
    if node.left is not None:
```

```
        queue.append(node.left)
```

```
    # Enqueue right child
```

```
    if node.right is not None:
```

```
        queue.append(node.right)
```

```
#Driver Program to test above function
```

```
root = Node(1)
```

```
root.left = Node(2)
```

```
root.right = Node(3)
```

```
root.left.left = Node(4)
```

```
root.left.right = Node(5)
```

```
print "Level Order Traversal of binary tree is -"
```

```
printLevelOrder(root)
```

```
#This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

C#

```
// Iterative Queue based C# program
```

```
// to do level order traversal
```

```
// of Binary Tree
```

```
using System;
```

```
using System.Collections.Generic;
```

```
/* Class to represent Tree node */
```

```

public class Node
{
    public int data;
    public Node left, right;

    public Node(int item)
    {
        data = item;
        left = null;
        right = null;
    }
}

/* Class to print Level Order Traversal */
public class BinaryTree
{
    Node root;

    /* Given a binary tree. Print
    its nodes in level order using
    array for implementing queue */
    void printLevelOrder()
    {
        Queue<Node> queue = new Queue<Node>();
        queue.Enqueue(root);
        while (queue.Count != 0)
        {
            /* poll() removes the present head.
            For more information on poll() visit
            http://www.tutorialspoint.com/java/util/linkedlist\_poll.htm */
            Node tempNode = queue.Dequeue();
            Console.Write(tempNode.data + " ");

            /*Enqueue left child */
            if (tempNode.left != null)
            {
                queue.Enqueue(tempNode.left);
            }

            /*Enqueue right child */
            if (tempNode.right != null)
            {
                queue.Enqueue(tempNode.right);
            }
        }
    }

    // Driver code
    public static void Main()
    {
        /* creating a binary tree and entering
        the nodes */
        BinaryTree tree_level = new BinaryTree();
        tree_level.root = new Node(1);
        tree_level.root.left = new Node(2);
        tree_level.root.right = new Node(3);
        tree_level.root.left.left = new Node(4);
    }
}

```



```
tree_level.root.left.right = new Node(5);

Console.WriteLine("Level order traversal " +
                  "of binary tree is - ");
tree_level.printLevelOrder();
}
}

/* This code contributed by PrinciRaj1992 */
```

Output:

```
Level order traversal of binary tree is -
1 2 3 4 5
```

Time Complexity: $O(n)$ where n is number of nodes in the binary tree

Space Complexity: $O(n)$ where n is number of nodes in the binary tree

Level Order Tree Traversal | GeeksforGeeks

**References:**

http://en.wikipedia.org/wiki/Breadth-first_traversal

Please write comments if you find any bug in the above programs/algorithms or other ways to solve the same problem.

Recommended Posts:

[Print a Binary Tree in Vertical Order | Set 3 \(Using Level Order Traversal\)](#)

[Flatten Binary Tree in order of Level Order Traversal](#)

[Insertion in n-ary tree in given order and Level order traversal](#)

[Given level order traversal of a Binary Tree, check if the Tree is a Min-Heap](#)

[Zig Zag Level order traversal of a tree using single queue](#)

[Density of Binary Tree using Level Order Traversal](#)

[Perfect Binary Tree Specific Level Order Traversal](#)

[Perfect Binary Tree Specific Level Order Traversal | Set 2](#)

[Check if the level order traversal of a Binary Tree results in a palindrome](#)

[Calculate height of Binary Tree using Inorder and Level Order Traversal](#)

[Check if the given array can represent Level Order Traversal of Binary Search Tree](#)

[General Tree \(Each node can have arbitrary number of children\) Level Order Traversal](#)

[Build Binary Tree from BST such that it's level order traversal prints sorted data](#)

---> [Connect Nodes at same Level \(Level Order Traversal\)](#)

[Construct BST from its given level order traversal](#)

Improved By : [shrikanth13](#), [princiraj1992](#), [rathbhupendra](#), [nidhi_biet](#), [SBVKrishna](#)

Article Tags : [Queue](#) [Tree](#) [Amazon](#) [Cisco](#) [D-E-Shaw](#) [Flipkart](#) [Microsoft](#) [Morgan Stanley](#) [Ola Cabs](#) [Payu](#)

[Qualcomm](#) [Samsung](#) [tree-level-order](#)

Practice Tags : [Flipkart](#) [Morgan Stanley](#) [Amazon](#) [Microsoft](#) [Samsung](#) [D-E-Shaw](#) [Ola Cabs](#) [Payu](#) [Cisco](#) [Qualcomm](#)

[Queue](#) [Tree](#)

☐ To-do ☐ DoneBased on **474** vote(s)[Feedback/ Suggest Improvement](#)[Add Notes](#)[Improve Article](#)

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

A computer science portal for geeks

5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 201305
feedback@geeksforgeeks.org

COMPANY

About Us
Careers
Privacy Policy
Contact Us

PRACTICE

Courses
Company-wise
Topic-wise
How to begin?

LEARN

Algorithms
Data Structures
Languages
CS Subjects
Video Tutorials

CONTRIBUTE

Write an Article
Write Interview Experience
Internships
Videos

@geeksforgeeks, Some rights reserved