Print Right View of a Binary Tree

Difficulty Level : Medium • Last Updated : 11 Jun, 2022

Given a Binary Tree, print Right view of it. Right view of a Binary Tree is set of nodes visible when tree is visited from Right side.

```
Right view of following tree is 1 3 7 8

          1
       /     \
     2         3
   /   \     /   \
  4     5   6     7
                    \
                     8
```

Recommended PracticeRight View of Binary TreeTry It!

The problem can be solved using simple recursive traversal. We can keep track of level of a node by passing a parameter to all recursive calls. The idea is to keep track of maximum level also. And traverse the tree in a manner that right subtree is visited before left subtree. Whenever we see a node whose level is more than maximum level so

implementation of this approach.

C++

```
// C++ program to print right view of Binary Tree
#include <bits/stdc++.h>
using namespace std;

struct Node
{
    int data;
    struct Node *left, *right;
};

// A utility function to
// create a new Binary Tree Node
struct Node *newNode(int item)
{
    struct Node *temp = (struct Node *)malloc(
                            sizeof(struct Node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

// Recursive function to print
// right view of a binary tree.
void rightViewUtil(struct Node *root,
                   int level, int *max_level)
{
    // Base Case
    if (root == NULL) return;

    // If this is the last Node of its level
    if (*max_level < level)
    {
        cout << root->data << "\t";
        *max_level = level;
    }
```

```
}

// A wrapper over rightViewUtil()
void rightView(struct Node *root)
{
    int max_level = 0;
    rightViewUtil(root, 1, &max_level);
}

// Driver Code
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);
    root->right->left = newNode(6);
    root->right->right = newNode(7);
    root->right->right->right = newNode(8);

    rightView(root);

    return 0;
}

// This code is contributed by SHUBHAMSINGH10
```

C

```
// C program to print right view of Binary Tree
#include<stdio.h>
#include<stdlib.h>

struct Node
{
    int data;
    struct Node *left, *right;
};
```

```
    struct Node *temp =  (struct Node *)malloc(sizeof(struct Node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

// Recursive function to print right view of a binary tree.
void rightViewUtil(struct Node *root, int level, int *max_level)
{
    // Base Case
    if (root==NULL)  return;

    // If this is the last Node of its level
    if (*max_level < level)
    {
        printf("%d\t", root->data);
        *max_level = level;
    }

    // Recur for right subtree first, then left subtree
    rightViewUtil(root->right, level+1, max_level);
    rightViewUtil(root->left, level+1, max_level);
}

// A wrapper over rightViewUtil()
void rightView(struct Node *root)
{
    int max_level = 0;
    rightViewUtil(root, 1, &max_level);
}

// 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);
    root->right->left = newNode(6);
    root->right->right = newNode(7);
```

```
    return 0;
}
```

Java

```
// Java program to print right view of binary tree

// A binary tree node
class Node {

    int data;
    Node left, right;

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

// class to access maximum level by reference
class Max_level {

    int max_level;
}

class BinaryTree {

    Node root;
    Max_level max = new Max_level();

    // Recursive function to print right view of a binary tree.
    void rightViewUtil(Node node, int level, Max_level max_level) {

        // Base Case
        if (node == null)
            return;

        // If this is the last Node of its level
        if (max_level.max_level < level) {
            System.out.print(node.data + " ");
```

```
        rightViewUtil(node.right, level + 1, max_level);
        rightViewUtil(node.left, level + 1, max_level);
    }

    void rightView()
    {
        rightView(root);
    }

    // A wrapper over rightViewUtil()
    void rightView(Node node) {

        rightViewUtil(node, 1, max);
    }

    // Driver program to test the 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);
        tree.root.right.left = new Node(6);
        tree.root.right.right = new Node(7);
        tree.root.right.left.right = new Node(8);

        tree.rightView();

        }
}

// This code has been contributed by Mayank Jaiswal
```

Python

```
# Python program to print right view of Binary Tree

# A binary tree node
class Node:
```

```
        self.right = None

# Recursive function to print right view of Binary Tree
# used max_level as reference list ..only max_level[0]
# is helpful to us
def rightViewUtil(root, level, max_level):

    # Base Case
    if root is None:
        return

    # If this is the last node of its level
    if (max_level[0] < level):
        print "%d   " %(root.data),
        max_level[0] = level

    # Recur for right subtree first, then left subtree
    rightViewUtil(root.right, level+1, max_level)
    rightViewUtil(root.left, level+1, max_level)

def rightView(root):
    max_level = [0]
    rightViewUtil(root, 1, max_level)


# 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)
root.right.left = Node(6)
root.right.right = Node(7)
root.right.left.right = Node(8)

rightView(root)

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

C#

```
// A binary tree node
public class Node
{

    public int data;
    public Node left, right;

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

// class to access maximum level by reference
public class Max_level
{

    public int max_level;
}

public class BinaryTree
{

    public Node root;
    public Max_level max = new Max_level();

    // Recursive function to print right view of a binary tree.
    public virtual void rightViewUtil(Node node, int level,
                                      Max_level max_level)
    {

        // Base Case
        if (node == null)
        {
            return;
        }

        // If this is the last Node of its level
        if (max level.max level < level)
```

```
        // Recur for right subtree first, then left subtree
        rightViewUtil(node.right, level + 1, max_level);
        rightViewUtil(node.left, level + 1, max_level);
    }

    public virtual void rightView()
    {
        rightView(root);
    }

    // A wrapper over rightViewUtil()
    public virtual void rightView(Node node)
    {

        rightViewUtil(node, 1, max);
    }

    // Driver program to test the 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);
        tree.root.right.left = new Node(6);
        tree.root.right.right = new Node(7);
        tree.root.right.left.right = new Node(8);

        tree.rightView();

    }
}

// This code is contributed by Shrikant13
```

Javascript

```
class Node
{
    constructor(item) {
       this.left = null;
       this.right = null;
       this.data = item;
    }
}

let max_level = 0;

let root;

// Recursive function to print right view of a binary tree.
function rightViewUtil(node, level) {

    // Base Case
    if (node == null)
        return;

    // If this is the last Node of its level
    if (max_level < level) {
        document.write(node.data + " ");
        max_level = level;
    }

    // Recur for right subtree first, then left subtree
    rightViewUtil(node.right, level + 1);
    rightViewUtil(node.left, level + 1);
}

function rightView()
{
    rightview(root);
}

// A wrapper over rightViewUtil()
function rightview(node) {

    rightViewUtil(node, 1);
```

```
    root.right = new Node(3);
    root.left.left = new Node(4);
    root.left.right = new Node(5);
    root.right.left = new Node(6);
    root.right.right = new Node(7);
    root.right.left.right = new Node(8);

    rightView();

</script>
```

Output

```
 1    3    7    8
```

Right view of Binary Tree using Queue

Time Complexity: The function does a simple traversal of the tree, so the complexity is O(n).

Auxiliary Space: O(n)

print the right most node of every level. So, we will do a level order traversal on the tree and print the last node at every level. Below is the implementation of above approach:

C++

```
// C++ program to print left view of
// Binary Tree

#include <bits/stdc++.h>
using namespace std;

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

// function to print Right view of
// binary tree
void printRightView(Node* root)
{
    if (root == NULL)
        return;

    queue<Node*> q;
    q.push(root);

    while (!q.empty()) {
        // get number of nodes for each level
```

```
            Node* x = q.front();
            q.pop();

            // print the last node of each level
            if (n == 0) {
                cout << x->data << " ";
            }
            // if left child is not null push it into the
            // queue
            if (x->left)
                q.push(x->left);
            // if right child is not null push it into the
            // queue
            if (x->right)
                q.push(x->right);
        }
    }
}

// Driver code
int main()
{
    // Let's construct the tree as
    // shown in example

    Node* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    root->right->left = newNode(6);
    root->right->right = newNode(7);
    root->right->left->right = newNode(8);

    printRightView(root);
}

// This code is contributed by
// Snehasish Dhar
```

```java
// JAVA program to print right view of
// Binary Tree

import java.io.*;
import java.util.LinkedList;
import java.util.Queue;

// A Binary Tree Node
class Node {
    int data;
    Node left, right;
    public Node(int d)
    {
        data = d;
        left = right = null;
    }
}

class BinaryTree {
    Node root;

    // function to print Right view of
    // binary tree
    void rightView(Node root)
    {
        if (root == null) {
            return;
        }

        Queue<Node> q = new LinkedList<>();
        q.add(root);

        while (!q.isEmpty()) {

            // get number of nodes for each level
            int n = q.size();

            // traverse all the nodes of the current level
            for (int i = 0; i < n; i++) {
                Node curr = q.peek();
                q.remove();
```

```
                    System.out.print(" ");
                }

                // if left child is not null add it into
                // the
                // queue
                if (curr.left != null) {
                    q.add(curr.left);
                }

                // if right child is not null add it into
                // the
                // queue
                if (curr.right != null) {
                    q.add(curr.right);
                }
            }
        }
    }

    // Driver code
    public static void main(String[] args)
    {

        // Let's construct the tree as
        // shown in example
        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);
        tree.root.right.left = new Node(6);
        tree.root.right.right = new Node(7);
        tree.root.right.left.right = new Node(8);

        tree.rightView(tree.root);
    }
}

// This code is contributed by Biswajit Rajak
```

```python
# Python3 program to print right
# view of Binary Tree
from collections import deque

# A binary tree node
class Node:

    # A constructor to create a new
    # Binary tree Node
    def __init__(self, val):
        self.data = val
        self.left = None
        self.right = None

# Function to print Right view of
# binary tree
def rightView(root):

    if root is None:
        return

    q = deque()
    q.append(root)

    while q:

        # Get number of nodes for each level
        n = len(q)

        # Traverse all the nodes of the
        # current level

        while n > 0:
            n -= 1

            # Get the front node in the queue
            node = q.popleft()

            # Print the last node of each level
            if n == 0:
                print(node.data, end = " ")
```

```
                q.append(node.left)

            # If right child is not null push
            # it into the queue
            if node.right:
                q.append(node.right)

# Driver code

# Let's construct the tree as
# shown in example
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
root.right.left = Node(6)
root.right.right = Node(7)
root.right.left.right = Node(8)

rightView(root)

# This code is contributed by Pulkit Pansari
```

C#

```
// C# program to print right view of
// Binary Tree
using System;
using System.Collections.Generic;

// A Binary Tree Node
public  class Node {
  public    int data;
  public Node left, right;
  public Node(int d)
  {
    data = d;
    left = right = null;
  }
```

```
// function to print Right view of
// binary tree
public  void rightView(Node root)
{
  if (root == null) {
    return;
  }

  Queue<Node > q = new Queue<Node>();
  q.Enqueue(root);

  while (q.Count!=0) {

    // get number of nodes for each level
    int n = q.Count;

    // traverse all the nodes of the current level
    for (int i = 0; i < n; i++) {
      Node curr = q.Peek();
      q.Dequeue();

      // print the last node of each level
      if (i == n - 1) {
        Console.Write(curr.data);
        Console.Write(" ");
      }

      // if left child is not null add it into
      // the
      // queue
      if (curr.left != null) {
        q.Enqueue(curr.left);
      }

      // if right child is not null add it into
      // the
      // queue
      if (curr.right != null) {
        q.Enqueue(curr.right);
      }
```

```
  // Driver Code
  public static void Main()
  {
    // Let's construct the tree as
    // shown in example
    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);
    tree.root.right.left = new Node(6);
    tree.root.right.right = new Node(7);
    tree.root.right.left.right = new Node(8);

    tree.rightView(tree.root);

  }
}

// This code is contributed by jana_sayantan.
```

Javascript

```
<script>

    // JavaScript program to print left view of Binary Tree

    class Node
    {
        constructor(data) {
           this.left = null;
           this.right = null;
           this.data = data;
        }
    }

    // Utility function to create a new tree node
    function newNode(data)
    {
```

```
// function to print Right view of
// binary tree
function printRightView(root)
{
    if (root == null)
        return;

    let q = [];
    q.push(root);

    while (q.length > 0) {
        // get number of nodes for each level
        let n = q.length;

        // traverse all the nodes of the current level
        while (n-- > 0) {

            let x = q[0];
            q.shift();

            // print the last node of each level
            if (n == 0) {
                document.write(x.data + " ");
            }
            // if left child is not null push it into the
            // queue
            if (x.left != null)
                q.push(x.left);
            // if right child is not null push it into the
            // queue
            if (x.right != null)
                q.push(x.right);
        }
    }
}

// Let's construct the tree as
// shown in example

let root = newNode(1);
root.left = newNode(2);
```

```
    root.right.right = newNode(7);
    root.right.left.right = newNode(8);

    printRightView(root);

</script>
```

Output

```
1 3 7 8
```

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

Auxiliary Space: O(n) since using auxiliary space for queue

This article is contributed by **Biswajit Rajak**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Like 47

Previous

Print Left View of a Binary Tree

Next

Right view of Binary Tree using Queue

RECOMMENDED ARTICLES

Page : 1 2 3

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Medium

Easy | Normal | Medium | Hard | Expert

Improved By : shrikanth13, SHUBHAMSINGH10, dsnehasish74, mukesh07, pansaripulkit13, divyeshrabadiya07, rajakbiswajit409, chauhanankur94, jana_sayantan, polymatir3j

Article Tags : Accolite, Adobe, Amazon, MakeMyTrip, Snapdeal, tree-view, Tree

Practice Tags : Accolite, Amazon, Snapdeal, MakeMyTrip, Adobe, Tree

Improve Article | Report Issue

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

Load Comments

GeeksforGeeks
A-143, 9th Floor, Sovereign Corporate Tower,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

NodeJS