GeeksforGeeks

A computer science portal for geeks

Placements Practice GATE CS IDE Q&A GeeksQuiz

Find next right node of a given key

Given a Binary tree and a key in the binary tree, find the node right to the given key. If there is no node on right side, then return NULL. Expected time complexity is O(n) where n is the number of nodes in the given binary tree.

For example, consider the following Binary Tree. Output for 2 is 6, output for 4 is 5. Output for 10, 6 and 5 is NULL.

```
                  10
               /      \
              2         6
           /   \         \
         8      4          5
```

We strongly recommend you to minimize the browser and try this yourself first.

Solution: The idea is to do level order traversal of given Binary Tree. When we find the given key, we just check if the next node in level order traversal is of same level, if yes, we return the next node, otherwise return NULL.

C++

```
/* Program to find next right of a given key */
#include <iostream>
#include <queue>
using namespace std;

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

// Method to find next right of given key k, it returns NULL if k is
// not present in tree or k is the rightmost node of its level
node* nextRight(node *root, int k)
```

```
{
    // Base Case
    if (root == NULL)
        return 0;

    // Create an empty queue for level order tarversal
    queue<node *> qn; // A queue to store node addresses
    queue<int> ql;   // Another queue to store node levels

    int level = 0;  // Initialize level as 0

    // Enqueue Root and its level
    qn.push(root);
    ql.push(level);

    // A standard BFS loop
    while (qn.size())
    {
        // dequeue an node from qn and its level from ql
        node *node = qn.front();
        level = ql.front();
        qn.pop();
        ql.pop();

        // If the dequeued node has the given key k
        if (node->key == k)
        {
            // If there are no more items in queue or given node is
            // the rightmost node of its level, then return NULL
            if (ql.size() == 0 || ql.front() != level)
               return NULL;

            // Otherwise return next node from queue of nodes
            return qn.front();
        }

        // Standard BFS steps: enqueue children of this node
        if (node->left != NULL)
        {
            qn.push(node->left);
            ql.push(level+1);
        }
        if (node->right != NULL)
        {
            qn.push(node->right);
            ql.push(level+1);
        }
    }

    // We reach here if given key x doesn't exist in tree
    return NULL;
}

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

// A utility function to test above functions
void test(node *root, int k)
{
    node *nr = nextRight(root, k);
    if (nr != NULL)
      cout << "Next Right of " << k << " is " << nr->key << endl;
    else
      cout << "No next right node found for " << k << endl;
}

// Driver program to test above functions
int main()
{
    // Let us create binary tree given in the above example
    node *root = newNode(10);
    root->left = newNode(2);
    root->right = newNode(6);
    root->right->right = newNode(5);
    root->left->left = newNode(8);
    root->left->right = newNode(4);

    test(root, 10);
    test(root, 2);
    test(root, 6);
    test(root, 5);
    test(root, 8);
    test(root, 4);
    return 0;
}
```

Java

```
// Java program to find sum of all numbers that are formed from root
// to leaf paths

import java.util.LinkedList;
import java.util.Queue;
```

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

    int data;
    Node left, right;

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

class BinaryTree {

    static Node root;

    // Method to find next right of given key k, it returns NULL if k is
    // not present in tree or k is the rightmost node of its level
    Node nextRight(Node first, int k) {

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

        // Create an empty queue for level order tarversal
        Queue<Node> qn = new LinkedList<Node>(); // A queue to store node addresses
        Queue<Integer> ql = new LinkedList<Integer>();   // Another queue to store node levels

        int level = 0;  // Initialize level as 0

        // Enqueue Root and its level
        qn.add(first);
        ql.add(level);

        // A standard BFS loop
        while (qn.size() != 0) {

            // dequeue an node from qn and its level from ql
            Node node = qn.peek();
            level = ql.peek();
            qn.remove();
            ql.remove();

            // If the dequeued node has the given key k
            if (node.data == k) {

                // If there are no more items in queue or given node is
                // the rightmost node of its level, then return NULL
                if (ql.size() == 0 || ql.peek() != level) {
                    return null;
                }

                // Otherwise return next node from queue of nodes
                return qn.peek();
            }

            // Standard BFS steps: enqueue children of this node
            if (node.left != null) {
                qn.add(node.left);
                ql.add(level + 1);
            }
            if (node.right != null) {
                qn.add(node.right);
                ql.add(level + 1);
            }
        }

        // We reach here if given key x doesn't exist in tree
        return null;
    }

    // A utility function to test above functions
    void test(Node node, int k) {
        Node nr = nextRight(root, k);
        if (nr != null) {
            System.out.println("Next Right of " + k + " is " + nr.data);
        } else {
            System.out.println("No next right node found for " + k);
        }
    }

    // driver program to test above functions
    public static void main(String args[]) {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(10);
        tree.root.left = new Node(2);
        tree.root.right = new Node(6);
        tree.root.right.right = new Node(5);
        tree.root.left.left = new Node(8);
        tree.root.left.right = new Node(4);

        tree.test(root, 10);
        tree.test(root, 2);
        tree.test(root, 6);
        tree.test(root, 5);
        tree.test(root, 8);
        tree.test(root, 4);
    }
}

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

Python

```
# Python program to find next right node of given key

# A Binary Tree Node
class Node:

    # Constructor to create a new node
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

# Method to find next right of a given key k, it returns
# None if k is not present in tree or k is the rightmost
# node of its level
def nextRight(root, k):

    # Base Case
    if root is None:
        return 0

    # Create an empty queue for level order traversal
    qn =  [] # A queue to store node addresses
    q1 = [] # Another queue to store node levels

    level = 0

    # Enqueue root and its level
    qn.append(root)
    q1.append(level)

    # Standard BFS loop
    while(len(qn) > 0):

        # Dequeu an node from qn and its level from q1
        node = qn.pop(0)
        level = q1.pop(0)

        # If the dequeued node has the given key k
        if node.key == k :

            # If there are no more items in queue or given
            # node is the rightmost node of its level,
            # then return None
            if (len(q1) == 0 or q1[0] != level):
                return None

            # Otherwise return next node from queue of nodes
            return qn[0]

        # Standard BFS steps: enqueue children of this node
        if node.left is not None:
            qn.append(node.left)
            q1.append(level+1)

        if node.right is not None:
            qn.append(node.right)
            q1.append(level+1)

    # We reach here if given key x doesn't exist in tree
    return None

def test(root, k):
    nr = nextRight(root, k)
    if nr is not None:
        print "Next Right of " + str(k) + " is " + str(nr.key)
    else:
        print "No next right node found for " + str(k)

# Driver program to test above function
root = Node(10)
root.left = Node(2)
root.right = Node(6)
root.right.right = Node(5)
root.left.left = Node(8)
root.left.right = Node(4)

test(root, 10)
test(root, 2)
test(root, 6)
test(root, 5)
test(root, 8)
test(root, 4)

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

Output:

```
No next right node found for 10
```

```
Next Right of 2 is 6
No next right node found for 6
No next right node found for 5
Next Right of 8 is 4
Next Right of 4 is 5
```

Time Complexity: The above code is a simple BFS traversal code which visits every enqueue and dequeues a node at most once. Therefore, the time complexity is O(n) where n is the number of nodes in the given binary tree.

Exercise: Write a function to find left node of a given node. If there is no node on the left side, then return NULL.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Spend less time debugging, more time coding.
Get Visual Studio Code FREE
Microsoft

85 Comments Category: Trees

Related Posts:

- Check if removing an edge can divide a Binary Tree in two halves
- Check sum of Covered and Uncovered nodes of Binary Tree
- Lowest Common Ancestor in a Binary Tree | Set 2 (Using Parent Pointer)
- Construct a Binary Search Tree from given postorder
- BFS vs DFS for Binary Tree
- Maximum difference between node and its ancestor in Binary Tree
- Inorder Non-threaded Binary Tree Traversal without Recursion or Stack
- Check if leaf traversal of two Binary Trees is same?

(Login to Rate and Mark)

2.6 Average Difficulty : **2.6/5.0**
Based on **6** vote(s)

☐ Add to TODO List
☐ Mark as DONE

Like Share 13 people like this.

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