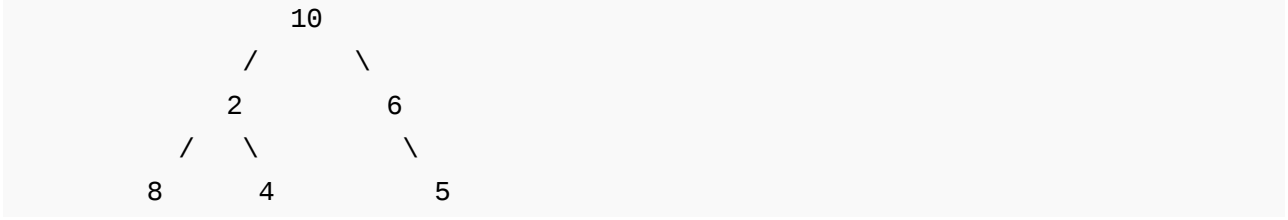
Find next right node of a given key

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.



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.

```
/* 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;
```


Interview Experiences
Advanced Data Structures
Dynamic Programming
Greedy Algorithms
Backtracking
Pattern Searching
Divide & Conquer
Mathematical Algorithms
Recursion

```
};

// 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);
        }
```


Popular Posts

All permutations of a given string
Memory Layout of C Programs
Understanding “extern” keyword in C
Median of two sorted arrays
Tree traversal without recursion and without stack!
Structure Member Alignment, Padding and Data Packing
Intersection point of two Linked Lists
Lowest Common Ancestor in a BST.
Check if a binary tree is BST or not
Sorted Linked List to Balanced BST

```
        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;
}
```

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

Recent Comments

affiszerv Your example has two 4s on row 3, that's why it...

Backtracking | Set 7 (Sudoku) · 27 minutes ago

RVM Can someone please elaborate this Qs from above...

Flipkart Interview | Set 6 · 47 minutes ago

Vishal Gupta I talked about as an Interviewer in general,...

Software Engineering Lab, Samsung Interview | Set 2 · 47 minutes ago

@meya Working solution for question 2 of 4f2f round....

Amazon Interview | Set 53 (For SDE-1) · 1 hour ago

sandeep void rearrange(struct node *head) {...

Given a linked list, reverse alternate nodes and append at the end · 2 hours ago

Neha I think that is what it should return as, in...

Find depth of the deepest odd level leaf node · 2 hours ago

Related Tpoics:

- Print a Binary Tree in Vertical Order | Set 2 (Hashmap based Method)
- Print Right View of a Binary Tree
- Red-Black Tree | Set 3 (Delete)
- Construct a tree from Inorder and Level order traversals
- Print all nodes at distance k from a given node
- Print a Binary Tree in Vertical Order | Set 1
- Interval Tree
- Check if a given Binary Tree is height balanced like a Red-Black Tree

Writing code in comment? Please use **ideone.com** and share the link here.

28 Comments GeeksforGeeks

Sort by Newest

Join the discussion…

VaraKalyan M • 2 months ago

I think, ql.size() == 0 ,, this should be qn.size()==0

• Reply • Share ›

sijayaraman • 2 months ago

Using one queue and one dummy node.

void print_right(struct node* root, int key)

```
{
queue<struct node*=""> myqueue;
struct node* dummy =newNode(-1);

myqueue.push(root);
myqueue.push(dummy);

while(!myqueue.empty())
{
struct node* current = myqueue.front();
myqueue.pop();
if(current->data == key)
{
if(myqueue.front()->data == -1)
{
cout<<"NULL"<<endl; return;="" }="" else="" {="" cout<<myqueue.front()-="">
```

see more

• Reply • Share ›

G Veera Sekhar • 2 months ago

```
public void findRight() {
List<node> list = new LinkedList<node>();
list.add(header);
findRight(list, 4);
}

private static void findRight(List<node> list, int k) {
if (list == null || list.isEmpty()) {
return;
}
int size = list.size();
for(int i=0;i<size;i++) {="" node="" root="list.remove(0);" if(root.data="=" k)="" {
```

right="list.remove(0);" system.out.println(right.data);="" }="" else="" {="" syster
break;="" }="" if(root.left="" !="null)" list.add(root.left);="" if(root.right="" !="null)"
findright(list,="" k);="" }="">

pulkit mehra • 3 months ago

It can be solved without using queues.
Just do a inorder traversal with 2 extra variables(level and flag, make them sta
Find the level of the node whose next right is to be determined, as soon as we
return
Find the next node with the same level and check if the flag is set. If we find a

naveenbobbili • 4 months ago

Solved using BFS with one queue

```
node* nextRight(node *root, int k)
{
if ((NULL != root) && (root->key != k)) {
queue<node*> q1;
q1.push(root);
while (!q1.empty()) {
int size = q1.size();
for (int i = 1; i <= size; i++) {
node* temp = q1.front();
if (temp->key == k) {
if (i == size)
return NULL;

q1.pop();
return q1.front();
} else {
```

see more

• Reply • Share ›

Coder011 • 4 months ago

A recursive approach to the question involves

(A) Finding the node with the given value (employing a simple BFS).
(B) Quitting the recursion and moving upwards to find the Next Right Node.

Link to Ideone : http://ideone.com/G15hoI

• Reply • Share ›

poonam Coder011 • a month ago

shd it not be dfs instead if bfs ...

• Reply • Share ›

Coder011 poonam • 25 days ago

we can use both, though i have used DFS (wrongly mentioned

• Reply • Share ›

Sumit Monga • 4 months ago

The recursive solution for this problem is:

```
void next_Right(struct node * root,int data,bool * is_found,int level,int * true_lev
//is_found checks whether the value whose right is to be located is found //leve
passed is 0)
//true_level is actual level of element whose right is to be located
//if target found store it in *temp.
{

if(!root)
```

return;

if(root->data == data)

{

*is_found = true;

*true_level = level;

see more

• Reply • Share ›

Hell • 4 months ago

I don't understand why output for 2 is not 4

• Reply • Share ›

guest ➚ Hell • 3 months ago

because the question is not the find the right node of the root..but the n just right to it

• Reply • Share ›

piyush bansal • 4 months ago

Hey,

Solved this with simple pre-order traversal without the usage of queues

void handleRightNodeToKey(struct BST* node, int refvalue, struct BST** resu

{

struct BST* temp = node;

if(temp != NULL)

{

if(temp->info == refvalue)

{

*resultnode = temp->right;

return;

see more

Reply • Share ›

groomnestle • 4 months ago

Level order traversal of a tree is actually a BFS on undirected graph (starting f
find the next node to be visited when given a node.

Reply • Share ›

maverick • 4 months ago

Here is a very simple recursive implementation, please have a look at it..

http://ideone.com/QuganS#stdin

Reply • Share ›

Geek123 • 4 months ago

Without Queue Implementation

#include<stdio.h>

struct TreeNode

{

struct TreeNode *left;

```
struct TreeNode *right;

int data;

};

void print(struct TreeNode *root)

{

int temp_data=0;
```

see more

• Reply • Share ›

Aniket Thakur • 4 months ago

Java code with output : http://opensourceforgeeks.blog...

• Reply • Share ›

Hara Shankar Nayak • 4 months ago

For exercise question,Use reverse pre-order traversal to get left node.
Thanks

• Reply • Share ›

Hara Shankar Nayak • 4 months ago

This question can be solve in O(1) space complexity.Use DFS instead of BFS
In DFS we can use pre-order traversal in which after getting the key,we will m
the key so that if we get any node with same level then that will be the first righ
dont want further nodes to be executed.
Please find the code below in C

```
#define null 0
typedef struct node
{
```

```
struct node *left;
struct node *right;
}Node;
Node *get_right_node_util(Node * root,int key,int level,int *k_lev)
{
if(root==null)
return null;
//if flag is true then it tells that the key is found and looking for right node
//k_lev gives the level of the key
```

see more

3 ▲ | ▼ • Reply • Share ›

Capablanca • 4 months ago

```
tree * NextNode(tree* root,int key) {
if(!root)
return root;

if(root->data==key)
return root->right;

tree* right= NextNode(root->right,key);
tree* left=NextNode(root->left,key);

if(right)
return right;
if(left)
return left;
return NULL;

}
```

▲ | ▼ • Reply • Share ›

xxmajia ➦ Capablanca • 4 months ago

seriously? can you please read the question first?

^ | v • Reply • Share ›

Sumit Poddar • 4 months ago

Please find the code without any space complexity and kindly let me know if th

```
public static int findRight(Node root, int k, int height, int p) {
if (root == null) {
return p;
}
if (root.data == k) {
return -height;
}
if (height == -(p)) {
return root.data;
}
p = findRight(root.left, k, height + 1, p);
p = findRight(root.right, k, height + 1, p);
return p;
}

public static int findLeft(Node root, int k, int height, int p) {
if (root == null) {
```

see more

^ | v • Reply • Share ›

viki • 4 months ago

Dear GFG,

This questions was asked in Microsoft coding test this year, and pointer to roo
was given.

```
struct TreeNode
{
int data;
struct TreeNode *left,*right, *parent;
};
```

3 ▲ | ▼ • Reply • Share ›

Kartik ➚ viki • 4 months ago

Viki, could you provide a solution for this problem. A simple solution is t
and so on.

▲ | ▼ • Reply • Share ›

Preeti ➚ Kartik • 4 months ago

```
getNextRIghtNode(struct node *key)
{
if(key == NULL)
return NULL;
struct node *par = key->parent;
if(par == NULL)//root has no parent
return NULL;
if(par && par->left == key && par->right != NULL)
return par->right;
struct node *x = getNextRightNode(par);
if(x == NULL)
return NULL;
if(x->left)
return x->left;
else if(x->right)
return x->right;
else
return NULL;
```

}

1 ^ | v • Reply • Share ›

Aditya Joshi ➚ Kartik • 4 months ago

Another simple solution is to first find the root. The root doesn't
BFS traversal.

3 ^ | v • Reply • Share ›

Kartik ➚ Aditya Joshi • 4 months ago

Thanks Aditya, this looks simple and better.

^ | v • Reply • Share ›

guest • 5 months ago

For the exercise part...Traverse the level from right to left instead of left to righ

And we can optimize the space complexity by using one queue instead of 2..

Just take a variable to keep track of how many nodes are pushed from x level
the nodes at x+1 level,If we found that node then if the level has more nodes re
else
return NULl

Same as iterative method to find the height of the tree....

3 ^ | v • Reply • Share ›

GFGFollower • 5 months ago

great to see a coding post after long time.

keep it up GFG :)

1 ^ | v • Reply • Share ›