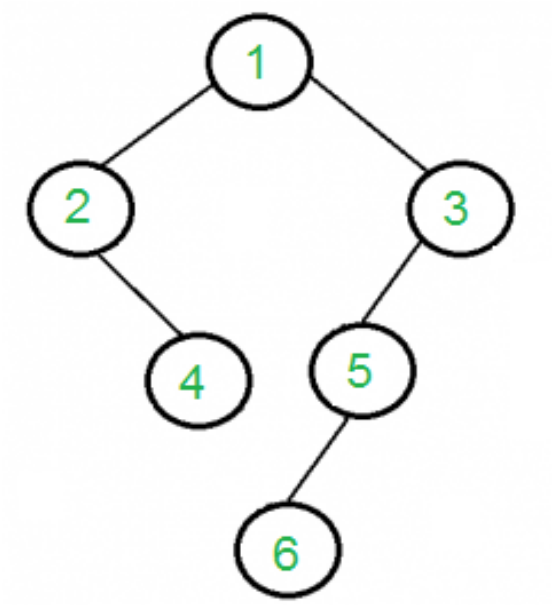
Print all nodes that don't have sibling

Given a Binary Tree, print all nodes that don't have a sibling (a sibling is a node that has same parent. In a Binary Tree, there can be at most one sibling). Root should not be printed as root cannot have a sibling.

For example, the output should be "4 5 6″ for the following tree.



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

This is a typical tree traversal question. We start from root and check if the node has one child, if yes then print the only child of that node. If node has both children, then recur for both the children.

```
/* Program to find singles in a given binary tree */
#include <iostream>
using namespace std;

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

// 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;
}

// Function to print all non-root nodes that don't have a sibling
void printSingles(struct node *root)
{
    // Base case
    if (root == NULL)
      return;

    // If this is an internal node, recur for left
    // and right subtrees
    if (root->left != NULL && root->right != NULL)
    {
        printSingles(root->left);
        printSingles(root->right);
    }

    // If left child is NULL and right is not, print right child
    // and recur for right child
    else if (root->right != NULL)
    {
        cout << root->right->key << " ";
        printSingles(root->right);
    }

    // If right child is NULL and left is not, print left child
    // and recur for left child
    else if (root->left != NULL)
    {
```


Popular Posts

```
        cout << root->left->key << " ";
        printSingles(root->left);
    }
}

// Driver program to test above functions
int main()
{
    // Let us create binary tree given in the above example
    node *root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->right = newNode(4);
    root->right->left = newNode(5);
    root->right->left->left = newNode(6);
    printSingles(root);
    return 0;
}
```

Output:

```
4 5 6
```

Time Complexity of above code is O(n) as the code does a simple tree traversal.

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

695 Subscribe

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.

15 Comments GeeksforGeeks

Sort by Newest

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

sujeet singh • 5 days ago

void print_nodes_without_siblings(struct node *btree)

{

if(!btree) return ;

if((btree->left) && (!btree->right))

cout << btree->left->data << "...";

else if((btree->right) && (!btree->left))

cout << btree->right->data << "...";

print_nodes_without_siblings(btree->left);

print_nodes_without_siblings(btree->right);

}

  • Reply • Share ›

anonymous • 20 days ago

What is wrong with this?

Print all nodes that do't have sibiling.

void printNode(Node root)

{

if(root==null)

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

return;

if(root.left!=null&&root.right==null)

S.O.P("root.left.key");

if(root.left==null&&root.right!=null)

S.O.P("root.right.key");

printNode(root.left);

printNode(root.right);

}

• Reply • Share ›

Zheng Luo • a month ago

Good solution

• Reply • Share ›

Gopi • 2 months ago

#include"tree.h"

void no_sibling(struct tree*root)

{

if(root == NULL)

return NULL;

if(root->left!=NULL && root->right == NULL)

{

```
printf("%d\t",root->left->data);

}

else if(root->left==NULL && root->right != NULL)

{
```

see more

Reply • Share ›

Mat • 2 months ago

Please find my try. Is anything wrong with below code?

```
public static void printNoSiblingNodes(Node n) {
if (n == null) return;
if(n.right == null && n.left != null) System.out.print(n.left.data + " ");
if(n.left == null && n.right != null) System.out.print(n.right.data + " ");
printNoSiblingNodes(n.left);
printNoSiblingNodes(n.right);
}
```

Reply • Share ›

Sankalp • 2 months ago

VEry simple question

Reply • Share ›

Pops • 2 months ago

Will this algorithms works if only root node present without left or right nodes?

Reply • Share ›

Guest • 2 months ago

Using Inorder traversal:

```
void tree::print_nodes_with_no_siblings(tree* root)

{

if (root == NULL)

return;

print_nodes_with_no_siblings(root->pLeft);

if (root->pLeft == NULL && root->pRight != NULL)

cout<<root->pRight->info<<endl; if="" (root-="">pLeft != NULL && root->pRigh

cout<<root->pLeft->info<<endl; print_nodes_with_no_siblings(root-="">pRight

}
```

• Reply • Share ›

Guest • 2 months ago

Using Inorder traversal:

```
void tree::print_nodes_with_no_siblings(tree* root)
{
if (root == NULL)
return;

print_nodes_with_no_siblings(root->pLeft);

if (root->pLeft == NULL && root->pRight != NULL)
cout<<root->pRight->info<<endl; if="" (root-="">pLeft != NULL && root->pRigh
cout<<root->pLeft->info<<endl; print_nodes_with_no_siblings(root-="">pRight
}
```

• Reply • Share ›

sah_ • 2 months ago

```
void printNoSiblings(node *head)
{
if(head)
{
if(head->lc && !(head->rc))
cout<<head->lc->data<<endl; else="" if(head-="">rc && !(head->lc))
cout<<head->rc->data<<endl; printnosiblings(head-="">lc);
printNoSiblings(head->rc);
}
}
```

• Reply • Share ›

agnb • 2 months ago

Even root node has no sibling.So, we should print it too.

• Reply • Share ›

agnb agnb • 2 months ago

I think the base condition of the function is wrong - it should be like belc

```
if(!root || (!root->left && !root->right))
return;
```

Please correct me if I am wrong.

• Reply • Share ›

Coder011 • 2 months ago

My attempt: http://ideone.com/F3bv6l

• Reply • Share ›

omar salem • 2 months ago

2 ^ | v • Reply • Share ›

FreaKode • 2 months ago

THis could be done using level order traversal ....and complexity will be same

1 ^ | v • Reply • Share ›
