Find depth of the deepest odd level leaf node

Write a C code to get the depth of the deepest odd level leaf node in a binary tree. Consider that level starts with 1. Depth of a leaf node is number of nodes on the path from root to leaf (including both leaf and root).

For example, consider the following tree. The deepest odd level node is the node with value 9 and depth of this node is 5.

```
        1
       /   \
     2      3
   /       /  \
  4       5    6
           \     \
            7     8
           /       \
          9         10
                    /
                   11
```

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

The idea is to recursively traverse the given binary tree and while traversing, maintain a variable “level” which will store the current node’s level in the tree. If current node is leaf then check “level” is odd or not. If level is odd then return it. If current node is not leaf, then recursively find maximum depth in left and right subtrees, and return maximum of the two depths.

C

```c
// C program to find depth of the deepest odd level leaf node
#include <stdio.h>
#include <stdlib.h>

// A utility function to find maximum of two integers
int max(int x, int y) { return (x > y)? x : y; }

// A Binary Tree node
```

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

// A utility function to allocate a new tree node
struct Node* newNode(int data)
{
    struct Node* node = (struct Node*) malloc(sizeof(struct Node));
    node->data = data;
    node->left = node->right = NULL;
    return node;
}

// A recursive function to find depth of the deepest odd level leaf
int depthOfOddLeafUtil(Node *root,int level)
{
    // Base Case
    if (root == NULL)
        return 0;

    // If this node is a leaf and its level is odd, return its level
    if (root->left==NULL && root->right==NULL && level&1)
        return level;

    // If not leaf, return the maximum value from left and right subtrees
    return max(depthOfOddLeafUtil(root->left, level+1),
               depthOfOddLeafUtil(root->right, level+1));
}

/* Main function which calculates the depth of deepest odd level leaf.
  This function mainly uses depthOfOddLeafUtil() */
int depthOfOddLeaf(struct Node *root)
{
    int level = 1, depth = 0;
    return depthOfOddLeafUtil(root, 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->right->left = newNode(5);
    root->right->right = newNode(6);
    root->right->left->right = newNode(7);
    root->right->right->right = newNode(8);
    root->right->left->right->left = newNode(9);
    root->right->right->right->right = newNode(10);
    root->right->right->right->right->left = newNode(11);

    printf("%d is the required depth\n", depthOfOddLeaf(root));
    getchar();
    return 0;
}
```

Run on IDE

Java

```java
// Java program to find depth of deepest odd level node
// A binary tree node
class Node {

    int data;
    Node left, right;

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

class BinaryTree {

    static Node root;

    // A recursive function to find depth of the deepest odd level leaf
    int depthOfOddLeafUtil(Node node, int level) {
        // Base Case
        if (node == null) {
            return 0;
        }

        // If this node is a leaf and its level is odd, return its level
        if (node.left == null && node.right == null && (level & 1) != 0) {
            return level;
        }

        // If not leaf, return the maximum value from left and right subtrees
        return Math.max(depthOfOddLeafUtil(node.left, level + 1),
                depthOfOddLeafUtil(node.right, level + 1));
    }

    /* Main function which calculates the depth of deepest odd level leaf.
     This function mainly uses depthOfOddLeafUtil() */
    int depthOfOddLeaf(Node node) {
        int level = 1, depth = 0;
        return depthOfOddLeafUtil(node, level);
    }

    public static void main(String args[]) {
        int k = 45;
        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.right.left = new Node(5);
        tree.root.right.right = new Node(6);
        tree.root.right.left.right = new Node(7);
        tree.root.right.right.right = new Node(8);
        tree.root.right.left.right.left = new Node(9);
        tree.root.right.right.right.right = new Node(10);
        tree.root.right.right.right.right.left = new Node(11);
        System.out.println(tree.depthOfOddLeaf(root) + " is the required depth");
    }
}

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

Run on IDE

Python

```
# Python program to find depth of the deepest odd level
# leaf node

# A Binary tree node
class Node:

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

# A recursive function to find depth of the deepest
# odd level leaf node
def depthOfOddLeafUtil(root, level):

    # Base Case
    if root is None:
        return 0

    # If this node is leaf and its level is odd, return
    # its level
    if root.left is None and root.right is None and level&1:
        return level

    # If not leaf, return the maximum value from left
    # and right subtrees
    return (max(depthOfOddLeafUtil(root.left, level+1),
                depthOfOddLeafUtil(root.right, level+1)))

# Main function which calculates the depth of deepest odd
# level leaf .
# This function mainly uses depthOfOddLeafUtil()
def depthOfOddLeaf(root):
    level = 1
    depth = 0
    return depthOfOddLeafUtil(root, level)

# Driver program to test above function
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.right.left = Node(5)
root.right.right = Node(6)
root.right.left.right = Node(7)
root.right.right.right = Node(8)
root.right.left.right.left = Node(9)
root.right.right.right.right = Node(10)
root.right.right.right.right.left= Node(11)

print "%d is the required depth" %(depthOfOddLeaf(root))

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

Run on IDE

Output:

```
5 is the required depth
```

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

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

65 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)

3 Average Difficulty : **3/5.0**
Based on **1** vote(s)

Add to TODO List

Mark as DONE

Like Share 22 people like this.

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

65 Comments **GeeksforGeeks** 1 **Login**

Recommend Share Sort by Newest

Join the discussion…

RAJ • 8 days ago

This condition should have an else for clarity -

// If this node is a leaf and its level is odd, return its level
if (root->left==NULL && root->right==NULL && level&1)
return level;

must be ,

if(root->left == NULL && root.right == NULL) {
if(level&1) {
return level;
} else {
return 0;
}
}

• Reply • Share ›

Mandy • 15 days ago

Similar approach, easy to understand

http://code.geeksforgeeks.org/...

• Reply • Share ›

shashank mishra • 21 days ago

it can be easily done by using level order traversal using stack..... in O(n) space.

• Reply • Share ›

Naveen Kumar • 2 months ago

There is no need of variable "depth" here !

• Reply • Share ›

Ramendu Shandilya • 7 months ago

//Iterative version of the same using level order traversal
public void deepestOddLevelLeaf(BSTNode root)

```
{
Queue<bstnode> q = new LinkedList<bstnode>();
q.add(root);
int size = 0 ;
int level = 1;
int compLevel = 0;
BSTNode deepestLeaf = null;
while(!q.isEmpty())
{
size = q.size();
if(size != 0 )
{
level++;
}
while(size != 0 )
{
```

see more

^ | ⌄ • Reply • Share ›

Abhishek Kumar • 7 months ago

SIMPLE AND EASY

```
#include<iostream>
using namespace std;
struct node
{
node* left;
node* right;
int data;
};
void check(node* root,int *prelevel,int* data,int level);
node * newNode(int data)
{
node* newnode=new node();
newnode->data=data;
newnode->left=NULL;
newnode->right=NULL;
return newnode;
}
```

see more

^ | ⌄ • Reply • Share ›

Rachit Nagdev • 8 months ago

it can also be done like this:

instead of recursing on children, recurse on grand children:

-if NULL is reached return 0
-if leaf is reached return 1
-compute odd leaf depths from grand children
-take max of these depths
-if max is 0 then no odd depth leaf exists so return 0
-otherwise return 2 + max

working code: ideone

@GeeksforGeeks

1 • Reply • Share ›

Deepak Sharma • 9 months ago

recursive implementation..

http://code.geeksforgeeks.org/...

• Reply • Share ›

Abhi • a year ago

GeeksforGeeks I thinks there is some thing wrong in function

depthOfOddLeafUtil(Node* root,int level)

it is not giving output of
struct Node* root = newNode(1);

root->left = newNode(2);

root->right = newNode(3);

the out should be 1
but it shows zero.

please let me know if I am wrong

• Reply • Share ›

guest Abhi • 10 months ago

it should be a leaf node..
1 isnt a leaf node

• Reply • Share ›

Abhi • a year ago

GeeksforGeeks...use simple recursive

#include<iostream>

#include<cstdio>

#include<cstdlib>

struct Node{

int data;

Node* left;

Node* right;

};

Node* newNode(int data1)

{

see more

^ | v • Reply • Share ›

thevagabond85 • a year ago

Alternatively, we can pass reference to a maxLevel var and update it recursively as in : http://ideone.com/CiANsa

^ | v • Reply • Share ›

thevagabond85 • a year ago

in function :
int depthOfOddLeaf(struct Node *root)
{
int level = 1, depth = 0;
return depthOfOddLeafUtil(root, level);
}

wth is depth=0 declared for?

^ | v • Reply • Share ›

AllergicToBitches • a year ago

```
int maxDepthUtil(struct node* node, int level, int &h)
{
   if (node==NULL)
       return 0;
   else
   {
        if(level&1 && level>h)
                h=level;
        maxDepthUtil(node->left, level+1, h);
        maxDepthUtil(node->right, level+1, h);
    }
}
int maxDepth(struct node* node)
{
        int h=0;
        maxDepthUtil(node, 0, h);
        return h;
}
```

1 • Reply • Share ›

Adwait S • a year ago

Gives both Odd level deepest as well as Even Level Deepest depending on flag value (1 = odd, 0 = even):

```
int deepestOddEvenLevelNode(struct TreeNode *root, int flag)
{
int x, y;

if(!root)
return 0;

x = deepestOddEvenLevelNode(root->left,1-flag);
y = deepestOddEvenLevelNode(root->right,1-flag);

if(x==0 && y==0 && flag)
return 1;
else if(x==0 && y==0 && !flag)
return 0;

if(x>y)
return x+1;
else
return y+1;
}
```

• Reply • Share ›

helper • a year ago

simple O(n) code in java: http://ideone.com/UScu1i

• Reply • Share ›

ryan • 2 years ago

```
void checkdepth(struct Node*root,struct Node**depthnode,int *maxdepth,int depth)
{
if(root==NULL)
return;
(*depth)++;
if(depth>(*maxdepth) &&depth%2==1 &&root->left==NULL&&root->right==NULL)
{
*depthnode=root;
*max=depth;
}
checkdepth(root->left,depthnode,maxdepth,depth);
checkdepth(root->right,depthnode,maxdepth,depth);
}
```

1 • Reply • Share ›

Guest • 2 years ago

Can also be done this way using iterative version of finding height in linear complexity: http://ideone.com/kGsgT5

• Reply • Share ›

vipinkaushal • 2 years ago

check it out it's working and simply a label order traversal

```
int findDepthOdd(node *root)
{
if(root==NULL)return 0;
queue<node*> q;
int l=1,d=0,s=0,i;
node *ptr=NULL;
q.push(root);
while(!q.empty())
{
s=q.size();
for(i=1;i<=s;i++)
{
ptr=q.front(); q.pop();
if(ptr->lchild==NULL&&ptr->rchild==NULL)
if(l>d&&l%2!=0)d=l;

if(ptr->lchild)q.push(ptr->lchild);
if(ptr->rchild)q.push(ptr->rchild);
}
l++;
}
return d;
}
```

• Reply • Share ›

DS+Algo • 2 years ago

@GeeksforGeeks
1) What is the use of depth variable in the function.
2)The function should be defined as follows otherwise if tree has only 2 levels, it will output 0 instead of 1:

```
int depthOfOddLeaf(struct Node *root)
{
int level = 1;
if(root==NULL)
return 0;

return max(level,depthOfOddLeafUtil(root, level));
}
```

• Reply • Share ›

thevagabond85 DS+Algo • a year ago

read question carefully : it asks to find deepest odd level LEAF.
For a tree having only 2 levels there is no odd level LEAVES hence the output = 0;

• Reply • Share ›

Guest • 2 years ago

This won't work when there are only first 2 levels. The output would be '0' but the expected output is '1'!
@GeeksforGeeks
Please Check it out...

• Reply • Share ›

vipinkaushal Guest • 2 years ago

the output should be 0
because we have to find

depth of the deepest odd level " leaf node " if there is no odd label leaf node then answer will be 0

• Reply • Share ›

Guest • 2 years ago

This won't work when there are only first 2 levels. The output would be '0' but the expected output is '1'! @GeeksforGeeks

Please Check it out

• Reply • Share ›

DS+Algo • 2 years ago

Easy way(Give your views):
calculate height of tree,
if(height is even) return height-1
else return height;

• Reply • Share ›

vipinkaushal DS+Algo • 2 years ago

no think carefully
consider tree with
inorder 2,1,6,4,7,3,8,5,9
preorder 1,2,3,4,6,7,5,8,9,
the answer should be 0 but according to you it's 3

• Reply • Share ›

DS+Algo vipinkaushal • 2 years ago

yeah, got it

• Reply • Share ›

Goku • 2 years ago

Using Level Order Traversal

http://ideone.com/ocZ0hw

• Reply • Share ›

<HoldOnLife!#> • 2 years ago

@GeeksforGeeks what is the use of int depth=0 ;

• Reply • Share ›

DS+Algo <HoldOnLife!#> • 2 years ago

yeah, reply me if u find out :)

• Reply • Share ›

Guest • 2 years ago

This won't work when there are only first 2 levels. The output would be '0' but the expected output is '1'!

1 • Reply • Share ›

instance • 2 years ago

void oddLeaf(class tree* root, int d, int& maxDepth) //oddLeaf(root, 1, 1);

```
{
if(!root)
return;
if(!(root->left) && !(root->right))
{
if(maxDepth == 1)
maxDepth = d;
else if((d > maxDepth) && (d%2))
maxDepth = d;
cout<<" "<<root->data<<" "<<d<<" "<<endl;="" d--;="" }="" oddleaf(root-="">left, d+1,
maxDepth);
oddLeaf(root->right, d+1, maxDepth);
}
```

^ | ⌄ • Reply • Share ›

samthebest • 2 years ago

```
int deepestOddNode(btnode *t,int level)
{
if(t!=NULL)
{ if(level%2!=0 && isleafNode(t))
return level;

return max(deepestOddNode(t->lchild,level+1),deepestOddNode(t->rchild,level+1));
}
else
return 0;
}
```

^ | ⌄ • Reply • Share ›

Mohaan Raja • 2 years ago

Why can't we just find the height of the tree and return the height if it is odd or return the height-1 if it is even.

```
#include <stdio.h>
#include <stdlib.h>

// A utility function to find maximum of two integers
int max(int x, int y) { return (x > y)? x : y; }

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

```
// A utility function to allocate a new tree node
struct Node* newNode(int data)
{
struct Node* node = (struct Node*) malloc(sizeof(struct Node));
```

see more

2 ^ | v • Reply • Share ›

samthebest ➦ Mohaan Raja • 2 years ago

what about the case when the height is even and there is no odd level leaf node..??
thn it should return 0
calculating height will give ryt answer only when ur tree ends as a leaf node at a odd level!

^ | v • Reply • Share ›

Mohaan Raja ➦ samthebest • 2 years ago

Thanks samthebest, for correcting me...

^ | v • Reply • Share ›

Uma Trika • 2 years ago

```
void depthUtil(struct Node *root, int h, int *max)

{

if(root == NULL)

return;

if((h>*max) && (h&1) && !root->left && !root->right)

{

*max = h;

printf("Node: %d is at %d\n", root->data,*max);

}

depthUtil(root->left, h+1, max);

depthUtil(root->right, h+1, max);

}
```

^ | v • Reply • Share ›

dazer • 2 years ago

```
void findoddnode(node *root,int count,int *c )

{

if(root== NULL) return;

if(*c < count && count%2 !=0 )

{
//for leaf node condition ..
if( root->left == NULL && root->right == NULL){

*c = count; //to keep track of level ..

}

}

findoddnode(root->left,count+1,c) ;

findoddnode(root->right,count+1,c);

}
```

• Reply • Share ›

Mukesh • 2 years ago

Solution has some bug with the complete binary tree on level 4. it returns 0.

Add this condition when we check left==null && right==null

```
if (root->left==NULL && root->right==NULL)
if(level&1)
return level;
else
return level-1;
```

1 • Reply • Share ›

Neha Mukesh • 2 years ago

I think that is what it should return as, in the questions its mentioned " get the depth of the deepest odd level leaf node in a binary tree"

And in your aforementioned case there is no odd level leaf node IMO

• Reply • Share ›

Tushar Patil • 2 years ago

Here is my solution:

```
int depthOfOddLeaf(mynode *root, int level)
{
static int depthOddLevel;
int l,r;

if(root == NULL)
return 0;

if(root->left == NULL && root->right == NULL)
{
if(level % 2)
{
if(depthOddLevel < level)
return (depthOddLevel = level);
}
}

l = depthOfOddLeaf(root->left, level+1);
r = depthOfOddLeaf(root->right, level+1);

return (l>r) ? l : r;
}
```

^ | v • Reply • Share ›

Birendra Kumar • 2 years ago

```
Static int s = 0;
Static int value=0;
Void oddleavemax(struct node* root,int level)
{
If(root==NULL)
Return;
If(root->left==root->right==NULL)
{
If(level%2!=0 && s<level) {="" s="level;" value="root-">data;
}
}
oddleavemax(root->left,level+1);
oddleavemax(root->right,level+1);
}
Void main()
{
oddleavemax(root,1);
Print("%d,%d",value,s);
}
```

^ | v • Reply • Share ›

din • 3 years ago
pls validate me....
• Reply • Share ›

din • 3 years ago
Using Level Order Traversal in Java

```
int deepoddleaf() {
int req = 0;
int level = 1;
int x =-1;
Node temp = root;
Queue<node> q = new LinkedList<node>();
q.add(temp);
q.add(null);
while (!q.isEmpty()) {
temp = q.remove();
if (temp == null) {
if (!q.isEmpty()) {
q.add(null);
}
level++;
}
```

see more

• Reply • Share ›

Coder • 3 years ago
How can we achieve this using Level Order Traversal or the iterative version
• Reply • Share ›

CoderKnowledge • 3 years ago
"The deepest odd level node is the node with value 9 and depth of this node is 5." --->

i think 11 is the deepest odd node if 9 is deepest node we should get 4 as depth right? please correct if i am wrong.
• Reply • Share ›

pradeep • 3 years ago
int depth (struct node *root)

{ if (root==NULL)

return 0;

else

return 1+max (depth(root->left),depth(root->right));

}

• Reply • Share ›

tgh • 3 years ago

can anybody tell me how to keep track of the level of the node???

• Reply • Share ›

rht • 3 years ago

Why shouldn't we go for level order traversing?????
Wouldn't that be easier????

• Reply • Share ›

Vijay Apurva • 3 years ago

//we can get level of the max odd element as well as value of maximum element

// if anything wrong please correct me

```
void oddroot(struct node * q ,int currlevel ,int *maxv,int *maxlevel){
if(q==NULL) return ;
if(q->left==NULL&&q->right==NULL&& (currlevel%2)!=0){

if(currlevel>*maxlevel){
*maxv=q->data;
*maxlevel=currlevel;
}

if(currlevel==*maxlevel)
*maxv=(q->data > *maxv)?(q->data):(*maxv) ;
}
oddroot(q->left,currlevel+1,maxv,maxlevel);
oddroot(q->right,currlevel+1,maxv,maxlevel);
}
```

//we can get level of the max odd element as well as value of maximum element

```
int maxv=root->data;
int maxlevel=1;
oddroot(root,1,&maxv,&maxlevel);
printf(" \n max odd value at level = %d and the value is...= %d \n",maxlevel,maxv);
```

// if anything wrong please correct me

• Reply • Share ›

Load more comments