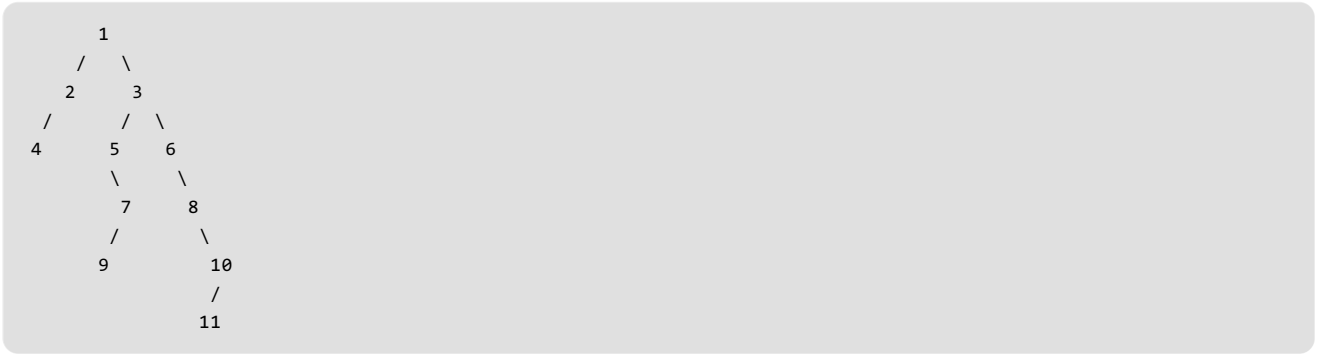
Find depth of the deepest odd level leaf node

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.



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 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;
}
```

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
{
    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(tree.root) +
                                                  " is the required depth");
    }
}

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

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)
```

Output:

```
5 is the required depth
```

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