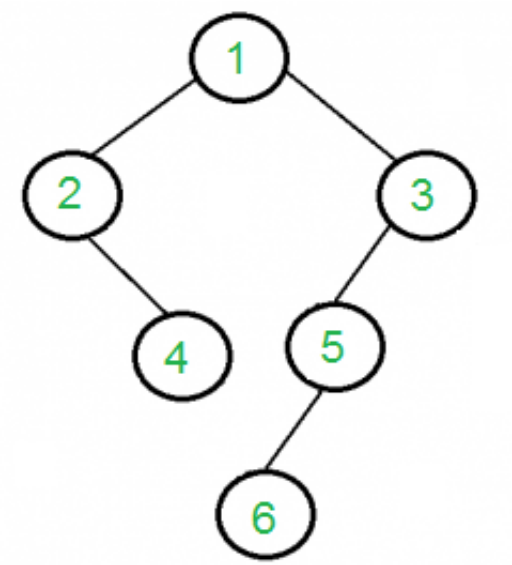
Print all nodes that don’t have sibling

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.

C++

```
/* 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)
    {
        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;
}
```

Java

```
// Java program to find distance between n1 and n2 using one traversal
// A binary tree node
class Node {

    int data;
    Node left, right;

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

class BinaryTree {

    static Node root;


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

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

    // If left child is NULL and right is not, print right child
    // and recur for right child
    else if (node.right != null)
    {
        System.out.print(node.right.data + " ");
        printSingles(node.right);
    }
```

```
    // If right child is NULL and left is not, print left child
    // and recur for left child
    else if (node.left != null)
    {
        System.out.print( node.left.data + " ");
        printSingles(node.left);
    }
}
    // Driver program to test the above functions
    public static void main(String args[]) {
        BinaryTree tree = new BinaryTree();

        /* Let us construct the tree shown in above diagram */
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.right = new Node(4);
        tree.root.right.left = new Node(5);
        tree.root.right.left.right = new Node(6);
        tree.printSingles(root);
    }
}

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

Python

```
# Program to find singles in a given binary tree

# A Binary Tree Node
class Node:

    # A constructor to create new tree node
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

# Function to print all non-root nodes that don't have
# a sibling
def printSingles(root):

    # Base Case
    if root is None:
        return

    # If this is an internal node , recur for left
```

```
    # and right subtrees
    if root.left is not None and root.right is not None:
        printSingles(root.left)
        printSingles(root.right)

    # If left child is NULL, and right is not, print
    # right child and recur for right child
    elif root.right is not None:
        print root.right.key,
        printSingles(root.right)

    # If right child is NULL and left is not, print
    # left child and recur for left child
    elif root.left is not None:
        print root.left.key,
        printSingles(root.left)

# Driver program to test above function
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.right = Node(4)
root.right.left = Node(5)
root.right.left.left = Node(6)
printSingles(root)

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

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

32 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 **3** vote(s)

Add to TODO List
Mark as DONE

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