Check if two nodes are cousins in a Binary Tree

Given the binary Tree and the two nodes say ‘a’ and ‘b’, determine whether the two nodes are cousins of each other or not.

Two nodes are cousins of each other if they are at same level and have different parents.

Example

```
     6
   /   \
  3     5
 / \   / \
7   8 1   3
Say two node be 7 and 1, result is TRUE.
Say two nodes are 3 and 5, result is FALSE.
Say two nodes are 7 and 5, result is FALSE.
```

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

The idea is to find level of one of the nodes. Using the found level, check if ‘a’ and ‘b’ are at this level. If ‘a’ and ‘b’ are at given level, then finally check if they are not children of same parent.

Following is the implementation of the above approach.

C

```
// C program to check if two Nodes in a binary tree are cousins
#include <stdio.h>
#include <stdlib.h>

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

// A utility function to create a new Binary Tree Node
struct Node *newNode(int item)
{
    struct Node *temp =  (struct Node *)malloc(sizeof(struct Node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

// Recursive function to check if two Nodes are siblings
int isSibling(struct Node *root, struct Node *a, struct Node *b)
{
    // Base case
    if (root==NULL)  return 0;

    return ((root->left==a && root->right==b)||
            (root->left==b && root->right==a)||
            isSibling(root->left, a, b)||
            isSibling(root->right, a, b));
}

// Recursive function to find level of Node 'ptr' in a binary tree
int level(struct Node *root, struct Node *ptr, int lev)
{
    // base cases
    if (root == NULL) return 0;
```

```
    if (root == ptr)  return lev;

    // Return level if Node is present in left subtree
    int l = level(root->left, ptr, lev+1);
    if (l != 0)  return l;

    // Else search in right subtree
    return level(root->right, ptr, lev+1);
}


// Returns 1 if a and b are cousins, otherwise 0
int isCousin(struct Node *root, struct Node *a, struct Node *b)
{
    //1. The two Nodes should be on the same level in the binary tree.
    //2. The two Nodes should not be siblings (means that they should
    // not have the same parent Node).
    if ((level(root,a,1) == level(root,b,1)) && !(isSibling(root,a,b)))
        return 1;
    else return 0;
}

// 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->left->right = newNode(5);
    root->left->right->right = newNode(15);
    root->right->left = newNode(6);
    root->right->right = newNode(7);
    root->right->left->right = newNode(8);

    struct Node *Node1,*Node2;
    Node1 = root->left->left;
    Node2 = root->right->right;

    isCousin(root,Node1,Node2)? puts("Yes"): puts("No");

    return 0;
}
```

Java

```
// Java program to check if two binary tree are cousins
class Node
{
    int data;
    Node left, right;

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

class BinaryTree
{
    Node root;

    // Recursive function to check if two Nodes are
    // siblings
    boolean isSibling(Node node, Node a, Node b)
    {
        // Base case
        if (node == null)
            return false;
```

```
        return ((node.left == a && node.right == b) ||
                (node.left == b && node.right == a) ||
                isSibling(node.left, a, b) ||
                isSibling(node.right, a, b));
    }

    // Recursive function to find level of Node 'ptr' in
    // a binary tree
    int level(Node node, Node ptr, int lev)
    {
        // base cases
        if (node == null)
            return 0;

        if (node == ptr)
            return lev;

        // Return level if Node is present in left subtree
        int l = level(node.left, ptr, lev + 1);
        if (l != 0)
            return l;

        // Else search in right subtree
        return level(node.right, ptr, lev + 1);
    }

    // Returns 1 if a and b are cousins, otherwise 0
    boolean isCousin(Node node, Node a, Node b)
    {
        // 1. The two Nodes should be on the same level
        //       in the binary tree.
        // 2. The two Nodes should not be siblings (means
        //    that they should not have the same parent
        //    Node).
        return ((level(node, a, 1) == level(node, b, 1)) &&
                (!isSibling(node, a, b)));
    }

    //Driver program to test above functions
    public static void main(String args[])
    {
        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.left.right = new Node(5);
        tree.root.left.right.right = new Node(15);
        tree.root.right.left = new Node(6);
        tree.root.right.right = new Node(7);
        tree.root.right.left.right = new Node(8);

        Node Node1, Node2;
        Node1 = tree.root.left.left;
        Node2 = tree.root.right.right;
        if (tree.isCousin(tree.root, Node1, Node2))
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}

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

Python

```
# Python program to check if two nodes in a binary
# tree are cousins
```

```
# A Binary Tree Node
class Node:

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

def isSibling(root, a , b):

    # Base Case
    if root is None:
        return 0

    return ((root.left == a and root.right ==b) or
            (root.left == b and root.right == a)or
            isSibling(root.left, a, b) or
            isSibling(root.right, a, b))

# Recursive function to find level of Node 'ptr' in
# a binary tree
def level(root, ptr, lev):

    # Base Case
    if root is None :
        return 0
    if root == ptr:
        return lev

    # Return level if Node is present in left subtree
    l = level(root.left, ptr, lev+1)
    if l != 0:
        return l

    # Else search in right subtree
    return level(root.right, ptr, lev+1)


# Returns 1 if a and b are cousins, otherwise 0
def isCousin(root,a, b):

    # 1. The two nodes should be on the same level in
    # the binary tree
    # The two nodes should not be siblings(means that
    # they should not have the smae parent node

    if ((level(root,a,1) == level(root, b, 1)) and
            not (isSibling(root, a, b))):
        return 1
    else:
        return 0


# Driver program to test above function
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
root.left.right.right = Node(15)
root.right.left = Node(6)
root.right.right = Node(7)
root.right.left.right = Node(8)

node1 = root.left.right
node2 = root.right.right

print "Yes" if isCousin(root, node1, node2) == 1 else "No"

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

Ouput:

```
Yes
```

Time Complexity of the above solution is O(n) as it does at most three traversals of binary tree.