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 {

    static Node root;
    // Recursive function to check if two Nodes are siblings

    boolean isSibling(Node node, Node node1, Node a, Node b) {

        // Base case
        if (node == null) {
            return false;
        }

        if ((node.left == a && node1.right == b)
                || (node.left == b && node1.right == a)) {
            return true;
        } else {
            return (isSibling(node.left, 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
    int 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).
        if ((level(node, a, 1) == level(node, b, 1)) && (isSibling(node, node, a, b))) {
            return 1;
        } else {
            return 0;
        }
    }

    //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(root, Node1, Node2) == 1) {
            System.out.println("Yes");
        } else {
            System.out.println("No");
        }
    }
}

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

Python

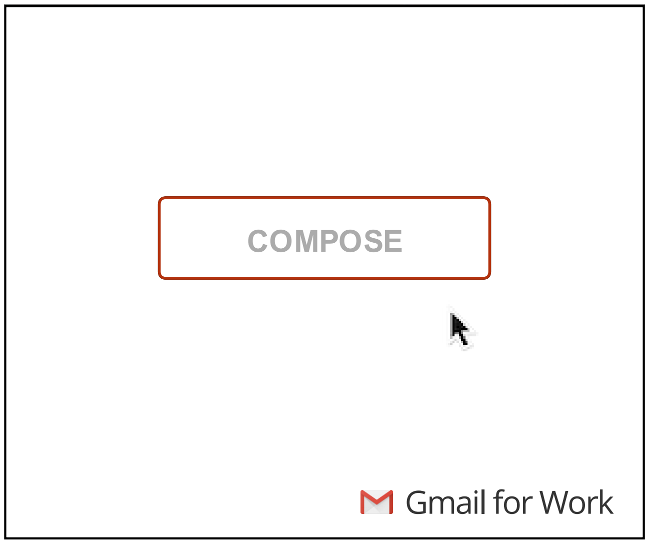
```
# Python program to check if two nodes ina binary tre are cousins

# 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.

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



Leave a comment 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 **6** vote(s)

☐ Add to TODO List
☐ Mark as DONE

Like Share 13 people like this.

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

72 Comments **GeeksforGeeks** 1 Login

Recommend 1 Share Sort by Newest

Join the discussion…

holdnet • 2 days ago

how to do this in one traversal ?

Reply • Share ›

Code_Addict • 11 days ago

Complete working code in java ::
1. 3 traversal of tree
2. O(n) Time Complexity
3. O(n) Space Complexity

public void isCousinNotsibling(Node root , Node n1 , Node n2)

{

int LCA= isCousinNotsiblingUtil(root, n1, n2);

// System.out.println("LCA == " + LCA);

if(LCA==-1)

{

System.out.println("nodes not found");

return;

see more

Reply • Share ›

Mandy • 12 days ago

Code above is not correct.

@GeeksForGeeks -
Please replace the sibling function with below function

```
boolean isSibling(Node node, Node a, Node b) {


        // Base case
        if (node == null) {
                return false;
        }
```

```
            boolean leftR = isSibling(node.left, a, b);
            boolean rightR = isSibling(node.right, a, b);

            if (node.data == a.data || node.data == b.data){
                    return true;
            }

            if(leftR && rightR)
                return false;
            else{
                return true;
            }
    }
```

• Reply • Share ›

Hiccup • 5 months ago

Only one traversal and in O(1)

algo outline

1. do preorder traversal

2. check their level and parent to decide whether they are cousin.

code link : https://app.box.com/s/hannhvp4...

1 • Reply • Share ›

nitin • 6 months ago

Done Using Level Order Traversal . Only one time traversal of the tree -

http://ideone.com/QHaRkv

If you find any issue / bug let me know .

• Reply • Share ›

Billionaire • 7 months ago

Like the article solution! Very clear!

1 • Reply • Share ›

Mandy Billionaire • 12 days ago

its incorrect try using 2 and 3

• Reply • Share ›

swayam raina • 7 months ago

find LCA
a,b should not be children of lca
levels should be same

is this approach correct???

• Reply • Share ›

Rachit Nagdev • 8 months ago
-find lca(root,a,b) in single traversal: O(n)
-check that both the children of lca are not a and b: O(1)

• Reply • Share ›

TulsiRam Rachit Nagdev • 8 months ago
galat bhai. . think again

• Reply • Share ›

swayam raina TulsiRam • 7 months ago
finding LCA
a,b should not be children of lca
levels should be same

is this correct???

• Reply • Share ›

sasha • 8 months ago
plz tell me the differnce between return 0 n return 1
like the way its used here

• Reply • Share ›

malhotra sasha • 8 months ago
it is similar to true and false..
0 means false..
1 means true.
in above code it returns 1 if all d conditions match "true"..else return 0 "false".

• Reply • Share ›

numid • 8 months ago
The complexity ca be further improved if the parent nodes of both the given nodes, and the level of both the nodes are calculated together in one recursive function.

• Reply • Share ›

NIKHIL SINGH • 8 months ago
if we pass a and b such as a=b,then it should not be cousin of itself but ur code will output it as yes it should be no

it as yes it should be no. ..
here u can easily check one condition for a!=b then it will be fine.
if(level(a)==level(b)&&!sibling(a,b)&&(a!=b)) then return 1 or true.
2 • Reply • Share ›

Deepak Sharma • 9 months ago
http://ideone.com/Op9fEX
• Reply • Share ›

Deepak Sharma • 9 months ago
using least common ancestor approach
• Reply • Share ›

Prakhar Agrawal • 9 months ago
#include<bits stdc++.h="">

using namespace std;

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));

see more

• Reply • Share ›

SiSi • 9 months ago
void isLevel(struct Node *root,int l,int k1,int k2,int &c)

{

if(root==NULL)

return;

if(l==1)

```
{
if(root->left && root->right)
{
if((root->left->data==k1 && root->right->data!=k2)||(root->left->data!=k1 && root->right->data==k2))
c++;
```

see more

• Reply • Share ›

vivek • a year ago

level order traversal jindabad!!!

3 • Reply • Share ›

Mr. Lazy • a year ago

Can be done using level order traversal!
http://ideone.com/dHaONR

6 • Reply • Share ›

lucy ➦ Mr. Lazy • 9 months ago

yes, if tree is complete binary tree then simple travel and find .... otherwise we have push in queue dummy value if left or right child not present in same level ,to this if both value in same level and not consecutive then 'TRUE' otherwise 'FALSE'..... is it right?

• Reply • Share ›

Mr. Lazy ➦ lucy • 9 months ago

Check the link ... I updated the comment :)

• Reply • Share ›

codemonk ➦ Mr. Lazy • 7 months ago

u are doing if(temp->right) then push temp->left in queue what if temp->left=NULL..

• Reply • Share ›

Mr. Lazy ➦ codemonk • 7 months ago

Oops.. that was some really nasty bug .. thanks dude!
changed to push left when left exists and push right when right exists. Thanks again :)

• Reply • Share ›

Rajnish Kr Jha • a year ago

A very simple approach:

#include<iostream>
#include<stdio.h>
#include<stdlib.h>
using namespace std;

struct node
{
int data;
struct node *left;
struct node *right;
};

struct node* newNode(int data)
{
struct node* node=(struct node*)malloc(sizeof(struct node));
node->data=data;
node->left=NULL;

see more

• Reply • Share ›

Ajitesh Mandal • a year ago

pls check and comment if this code is wrong..

http://ideone.com/hhZEq3

• Reply • Share ›

The_Geek • a year ago

Can be managed simply like this.

```
void cousins(Node* root, Node *parent,Node*& p1, Node*& p2, int &l1, int &l2, int lvl,
{
if(!root)
return;

if(root->key==n1)
{
p1=parent;
l1=lvl;
return;
}
```

```
if(root->key==n2)
{
p2=parent;
l2=lvl;
```

see more

5 ^ | v • Reply • Share ›

De • a year ago
Can we use the following method?
1. Find LCAs of both of them.
2. If the length of the LCA array is the same for both and the element just before the given keys(i.e. their parents) are different, they are cousins.

Please reply!

^ | v • Reply • Share ›

Mohit ➦ De • a year ago
Correct!! :)
^ | v • Reply • Share ›

Anonymous • a year ago
Cousin can be determined in single call. In that we calculate level of both node and also store parent of one node and check later that it parent for both.
^ | v • Reply • Share ›

shan • a year ago
can be solved by LCA method

http://ideone.com/1iOrli
^ | v • Reply • Share ›

Abhinav Bhardwaj • a year ago
http://ideone.com/oh0kIU
^ | v • Reply • Share ›

sparco • a year ago
//if found=4 then they are cousins and not siblings

void isCousin(struct Node* root, struct Node* a, struct node* b, int level,int *found,int *levelofnode1) {

if (!root) return false;

if(a==b) //this check can be done in a util function. need not be done everytime

return true;

if (root==a || root=b) {

if (found==0) { // a or b is found for first time

found=1;

levelofnode1=level;

} else if (found==1) { // a or b is found for second time

see more

• Reply • Share ›

Guest • a year ago

void isCousin(struct Node* root, struct Node* a, struct node* b, int level,int *found,int *levelofnode1) {

if (!root) return false;

if (root==a || root=b) {

if (found==0) { // a or b is found for first time

found=1;

levelofnode1=level;

if(a==b) //this is done so that we know it exists in the tree.

return true;

} else if (found==1) { // a or b is found for second time

if (levelofnode1==level)

see more

• Reply • Share ›

sasuke • a year ago

level of the two nodes can be compared in one single call
rather than calling level two times for both the nodes and then comparing the levels returned by the function.
int compare_level(struct Node *root, struct Node *a, struct Node *b,int *visited,int lev,int

```
*act)
{
// base cases
if (root == NULL) return 0;
if (root == a||root==b)
{
if(*visited == 0)
{
*act = lev;
*visited = 1;
}
else
{

if((*act)==lev)return 1;
else
return 0;
}
}
int x = compare_level(root->left,a,b,visited,lev+1,act);
if(x==1)return x;
return compare_level(root->right,a,b,visited,lev+1,act);
}
```

1 • Reply • Share ›

Raja • 2 years ago

Two nodes need not be cousins if they are at the same level and not have same parents .. that is , their great grandparent can be their LCA instead of grandparent

2 • Reply • Share ›

Vãîbhåv Joshî • 2 years ago

http://ideone.com/dUrurS

• Reply • Share ›

Karshit Jaiswal • 2 years ago

http://ideone.com/UK3mmP

Recursive approach using level order traversal.

• Reply • Share ›

Satya Swaroop ➦ Karshit Jaiswal • 2 years ago

This approach is iterative approach not recursive and it has some errors like count has to be set to zero after each level.

• Reply • Share ›

rahuljain1311 ➦ Satya Swaroop • 2 years ago
I dont think count needs to be set to zero after each level as
consider
if count updates only once at a level then if null of that level comes it will return false
if count updates twice then as the null comes it will return true
if not updated then it will remain zero, so NO problem!
Reply • Share ›

Satya Swaroop ➦ rahuljain1311 • 2 years ago
how do you tell that both the nodes are at same level? if count is 2
Reply • Share ›

rahuljain1311 ➦ Satya Swaroop • 2 years ago
Becoz in this approach @karshit is pushing NULL at the end of every level. So when the NULL comes he can say level has just finished
Reply • Share ›

Suman • 2 years ago
Java code http://ideone.com/fBluZb. We may not need recursive function isSibling() if we can keep track of the reference of parents.
1 • Reply • Share ›

INDRANIL • 2 years ago
the concept of solving the sum is good
Reply • Share ›

Vibhu Jawa • 2 years ago
We can just do level order transition and add NULL to mark the starting of each level and during level transition add this check for siblings , it saves us the trouble of checking level for each node and also the trouble of checking sibling in a different loop.

i have checked it and it works but am in doubt about complexity of the whole thing.

```
if(temp->left&&temp->right)
{
if((temp->left->data==p)&&(temp->right->data==q)||(temp->left->data==q)&&(temp->right->data==p))
{
cout<<"siblings";
return ;
}
}
```

• Reply • Share ›

Pratik • 2 years ago

Shouldn't a level order traversal be better for this? That would eliminate the check for same level and all we need to check is if they are from the same parent

2 • Reply • Share ›

Ankit Goyal • 2 years ago

What would be the best solution if its a Binary Search Tree instead of a simple Binary Tree?

• Reply • Share ›

satya • 2 years ago

public static void main(String[] args) throws Exception

{

//if(isUniqueString("sdfghjf"))

// System.out.println("it is unique string");

TreeNode rootTree = constructTree();

int value1 = 7, value2 = 10;

if(rootTree==null)

System.out.println("Given nodes are not cousins");

if(rootTree.data == value1 || rootTree.data == value2)

System.out.println("Given nodes are not cousins");

LevelClass levelClass = new LevelClass();

see more

• Reply • Share ›

GOPI GOPINATH • 2 years ago

A better approach would be storing the parent nodes while finding the level of the nodes. And compare the parents after ensuring that both nodes are in same level

4 • Reply • Share ›

Load more comments