Check if removing an edge can divide a Binary Tree in two halves

Given a Binary Tree, find if there exist edge whose removal creates two trees of equal size.

Examples:

```
Input : root of following tree
           5
         /   \
       1      6
      /      /  \
     3      7    4
Output : true
Removing edge 5-6 creates two trees of equal size


Input : root of following tree
           5
         /   \
       1      6
            /  \
           7    4
         /  \    \
        3    2    8
Output : false
There is no edge whose removal creates two trees
of equal size.
```

Source- Kshitij IIT KGP

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

Method 1 (Simple)

First count number of nodes in whole tree. Let count of all nodes be n. Now traverse tree and for every node,

find size of subtree rooted with this node. Let the subtree size be s. If n-s is equal to s, then return true, else false.

```
// C++ program to check if there exist an edge whose
// removal creates two trees of same size
#include<bits/stdc++.h>
using namespace std;

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

// utility function to create a new node
struct Node* newNode(int x)
{
    struct Node* temp = new Node;
    temp->data = x;
    temp->left = temp->right = NULL;
    return temp;
};

// To calculate size of tree with given root
int count(Node* root)
{
    if (root==NULL)
        return 0;
    return count(root->left) + count(root->right) + 1;
}

// This function returns true if there is an edge
// whose removal can divide the tree in two halves
// n is size of tree
bool checkRec(Node* root, int n)
{
    // Base cases
    if (root ==NULL)
       return false;

    // Check for root
    if (count(root) == n-count(root))
        return true;

    // Check for rest of the nodes
    return checkRec(root->left, n) ||
           checkRec(root->right, n);
}

// This function mainly uses checkRec()
```

```
bool check(Node *root)
{
    // Count total nodes in given tree
    int n = count(root);

    // Now recursively check all nodes
    return checkRec(root, n);
}

// Driver code
int main()
{
    struct Node* root = newNode(5);
    root->left = newNode(1);
    root->right = newNode(6);
    root->left->left = newNode(3);
    root->right->left = newNode(7);
    root->right->right = newNode(4);

    check(root)?  printf("YES") : printf("NO");

    return 0;
}
```

Output :

```
YES
```

Time complexity of above solution is $O(n^2)$ where n is number of nodes in given Binary Tree.

Method 2 (Efficient)

We can find the solution in O(n) time. The idea is to traverse tree in bottom up manner and while traversing keep updating size and keep checking if there is a node that follows the required property.

Below is C++ implementation of above idea.

```
// C++ program to check if there exist an edge whose
// removal creates two trees of same size
#include<bits/stdc++.h>
using namespace std;

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

```
// utility function to create a new node
struct Node* newNode(int x)
{
    struct Node* temp = new Node;
    temp->data = x;
    temp->left = temp->right = NULL;
    return temp;
};

// To calculate size of tree with given root
int count(Node* root)
{
    if (root==NULL)
        return 0;
    return count(root->left) + count(root->right) + 1;
}

// This function returns size of tree rooted with given
// root. It also set "res" as true if there is an edge
// whose removal divides tree in two halves.
// n is size of tree
int checkRec(Node* root, int n, bool &res)
{
    // Base case
    if (root == NULL)
       return 0;

    // Compute sizes of left and right children
    int c = checkRec(root->left, n, res) + 1 +
            checkRec(root->right, n, res);

    // If required property is true for current node
    // set "res" as true
    if (c == n-c)
        res = true;

    // Return size
    return c;
}

// This function mainly uses checkRec()
bool check(Node *root)
{
    // Count total nodes in given tree
    int n = count(root);

    // Initialize result and recursively check all nodes
    bool res = false;
    checkRec(root, n,  res);
```

```
    return res;
}

// Driver code
int main()
{
    struct Node* root = newNode(5);
    root->left = newNode(1);
    root->right = newNode(6);
    root->left->left = newNode(3);
    root->right->left = newNode(7);
    root->right->right = newNode(4);

    check(root)?  printf("YES") : printf("NO");

    return 0;
}
```

Output :

```
YES
```

This article is contributed by **Asaad Akram**. 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 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?
- Closest leaf to a given node in Binary Tree

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