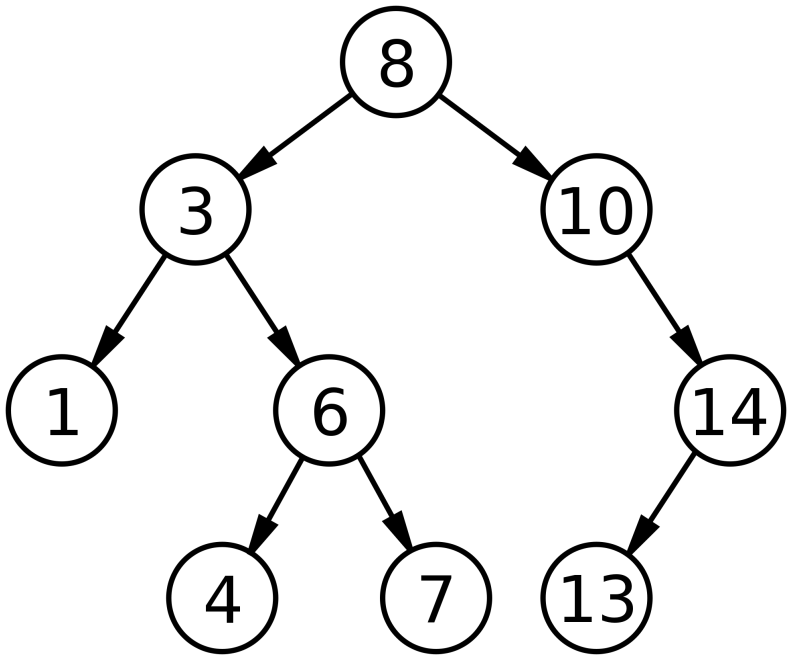
Check sum of Covered and Uncovered nodes of Binary Tree

Given a binary tree, you need to check whether sum of all covered elements is equal to sum of all uncovered elements or not.

In a binary tree, a node is called Uncovered if it appears either on left boundary or right boundary. Rest of the nodes are called covered.

For example, consider below binary tree




```
In above binary tree,
Covered node:    6, 4, 7
Uncovered node:   8, 3, 1, 10, 14, 13

The output for this tree should be false as
sum of covered and uncovered node is not same
```

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

For calculating sum of Uncovered nodes we will follow below steps:
1) Start from root, go to left and keep going until left child is available, if not go to right child and again follow same procedure until you reach a leaf node.

2) After step 1 sum of left boundary will be stored, now for right part again do the same procedure but now keep going to right until right child is available, if not then go to left child and follow same procedure until you reach a leaf node.

After above 2 steps sum of all Uncovered node will be stored, we can subtract it from total sum and get sum of covered elements and check for equines of binary tree.

```
// C++ program to find sum of Covered and Uncovered node of
// binary tree
#include <bits/stdc++.h>
using namespace std;

/* A binary tree node has key, pointer to left
   child and a pointer to right child */
struct Node
{
    int key;
    struct Node* left, *right;
};

/* To create a newNode of tree and return pointer */
struct Node* newNode(int key)
{
    Node* temp = new Node;
    temp->key = key;
    temp->left = temp->right = NULL;
    return (temp);
}

/* Utility function to calculate sum of all node of tree */
int sum(Node* t)
{
    if (t == NULL)
        return 0;
    return t->key + sum(t->left) + sum(t->right);
}

/* Recursive function to calculate sum of left boundary
   elements  */
int uncoveredSumLeft(Node* t)
{
    /*  If left node, then just return its key value   */
    if (t->left == NULL && t->right == NULL)
        return t->key;

    /*  If left is available then go left otherwise go right  */
    if (t->left != NULL)
        return t->key + uncoveredSumLeft(t->left);
    else
        return t->key + uncoveredSumLeft(t->right);
}

/* Recursive function to calculate sum of right boundary
   elements  */
int uncoveredSumRight(Node* t)
```

```
{
    /*  If left node, then just return its key value   */
    if (t->left == NULL && t->right == NULL)
        return t->key;

    /*  If right is available then go right otherwise go left  */
    if (t->right != NULL)
        return t->key + uncoveredSumRight(t->right);
    else
        return t->key + uncoveredSumRight(t->left);
}

// Returns sum of uncovered elements
int uncoverSum(Node* t)
{
    /* Initializing with 0 in case we don't have
       left or right boundary  */
    int lb = 0, rb = 0;

    if (t->left != NULL)
        lb = uncoveredSumLeft(t->left);
    if (t->right != NULL)
        rb = uncoveredSumRight(t->right);

    /* returning sum of root node, left boundry
       and right boundry*/
    return t->key + lb + rb;
}

// Returns true if sum of covered and uncovered elements
// is same.
bool isSumSame(Node *root)
{
    // Sum of uncovered elements
    int sumUC = uncoverSum(root);

    // Sum of all elements
    int sumT = sum(root);

    // Check if sum of covered and uncovered is same
    return (sumUC == (sumT - sumUC));
}

/* Helper function to print inorder traversal of
   binary tree   */
void inorder(Node* root)
{
    if (root)
    {
        inorder(root->left);
        printf("%d ", root->key);
        inorder(root->right);
    }
}

// Driver program to test above functions
int main()
{
    // Making above given diagram's binary tree
    Node* root = newNode(8);
    root->left = newNode(3);

    root->left->left = newNode(1);
    root->left->right = newNode(6);
    root->left->right->left = newNode(4);
```

```
    root->left->right->right = newNode(7);

    root->right = newNode(10);
    root->right->right = newNode(14);
    root->right->right->left = newNode(13);

    if (isSumSame(root))
        printf("Sum of covered and uncovered is same\n");
    else
        printf("Sum of covered and uncovered is not same\n");
}
```

Run on IDE

Output :

```
Sum of covered and uncovered is not same
```

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

7 Comments Category: Trees

Related Posts:

- 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
- Locking and Unlocking of Resources arranged in the form of n-ary Tree

(Login to Rate and Mark)

3 Average Difficulty : **3/5.0**
Based on **3** vote(s)

Add to TODO List

Mark as DONE

Like Share 6 people like this.

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

7 Comments GeeksforGeeks 1 Login

Recommend Share Sort by Newest

Join the discussion…

varun • 3 days ago
Java implementation:
http://ideone.com/TDw64G
• Reply • Share ›

MAXX • 5 days ago
Isn't 4 comes in the left boundary i.e. Left view of the tree.

Left view or left boundary of a tree consists of 1st node of all the levels of a tree;
• Reply • Share ›

Alexander Ivanov • 9 days ago
level order traversal ..
• Reply • Share ›

Suman Alexander Ivanov • 8 days ago
can you please explain how level order traversal can be used in this case?
• Reply • Share ›

Alexander Ivanov Suman • 8 days ago
Well. As far as I understood the task, all uncovered nodes are nodes that situated on the sides, and all covered nodes are nodes that are covered from both sides.
The idea is just make a level order traversal, adding the last the the first node and subtracting all other nodes.

• Reply • Share ›

.NetGeek ➚ Alexander Ivanov • 7 days ago

I even thought about this but It looks like the logic wont work. Just refer above tree and consider the last level, as per our logic 4 & 13 would be uncovered nodes and 7 would be covered node. However, 4 is a covered node.

• Reply • Share ›

Alexander Ivanov ➚ .NetGeek • 7 days ago

Yeah, you are right. I had to look more precisely,
Thank you

• Reply • Share ›