Code Wrestling Follow
May 24, 2021 · 5 min read · Listen

Branch Sum in Binary Tree — Competitive Programming with Time and Space Complexity

Question — 7(Level — Easy)

Welcome to the competitive series of ***Code Wrestling****. Today, we are gonna solve an easy problem, Branch Sums in Binary Tree.*

We will find the ***Time and Space complexity*** *of our solution and will also guide you to the process of finding an* ***optimized solution****.*

Visit our YouTube channel for more interesting stuff **Code Wrestling.**

Github Repository Link: Github/ClosestValueInBST

Go to the previous question → Question — 5 (Non-Constructible Change)

Code Wrestling

We are here to hustle with the complexity of machine learning and various other new technologies. We as a team try to…

www.youtube.com

Question

> *Write a function that takes in a Binary Tree and returns a list of its branch sums ordered from leftmost branch sum to rightmost branch sum.*
>
> *A branch sum is the sum of all values in a Binary Tree branch. A Binary Tree branch is a path of nodes in a tree that starts at the root node and ends at any leaf node. Each BinaryTree node has an integer value, a left child node, and a right child node. Children nodes can either be BinaryTree nodes themselves or None / null.*

Sample Input:

Sign In

Get started

```
tree =       1
          /     \
         2        3
       /   \    /   \
      4     5  6     7
    /   \  /
   8     9 10
```

Sample Output:

```
[15, 16, 18, 10, 11]

// 15 == 1 + 2 + 4 + 8
// 16 == 1 + 2 + 4 + 9
// 18 == 1 + 2 + 5 + 10
// 10 == 1 + 3 + 6
// 11 == 1 + 3 + 7
```

Solution

The main objective here is to find the sum of each branch. Since we have to find the sum of each branch, we can solve this problem by Recursion.

Approach:

First, understand how a Binary tree looks. Below is the code for a Binary Tree class.

```
public class BinaryTree
{
        public BinaryTree(int value)
        {
            this.value = value;
            this.left = null;
            this.right = null;
        }

        public int value;
        public BinaryTree left;
```

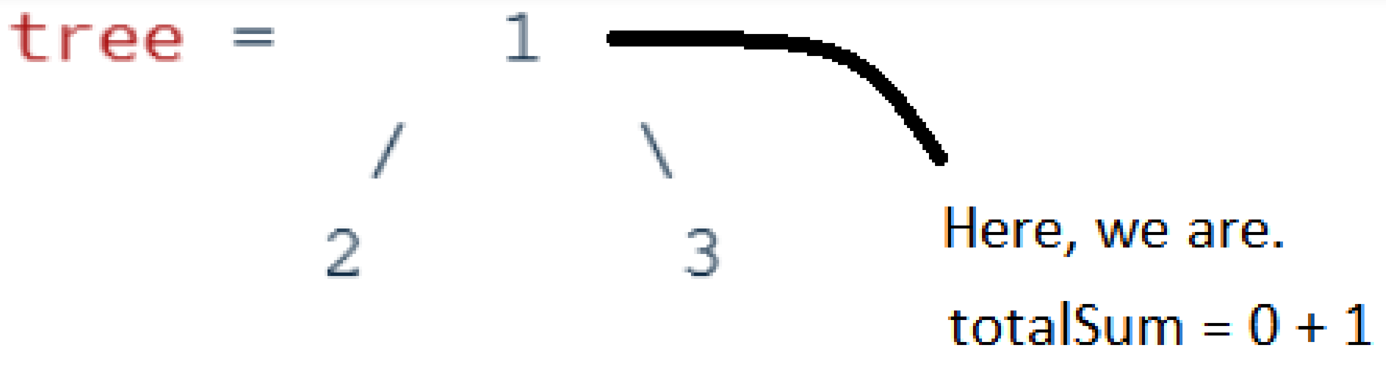
Create a list that stores the sum of branches, and then a variable total sum, that tracks the current sum, and at any point, if node == null, then return.

```
private void GetBranchSum(BinaryTree tree, int runningSum, List<int>
branchSum)
{
     if (tree == null) return;
     int totalSum = runningSum + tree.value;
}
```

So call the above function, the first time as:

```
List<int> branchSum = new List<int>();
GetBranchSum(tree, 0, branchSum)
```

Initially, when we traverse our tree:



Now we need to find the sum for the left node and then the right node, but before we do it, we have to find a base condition for our recursion function. So let say if we found the total sum, then we have to store it in the list.

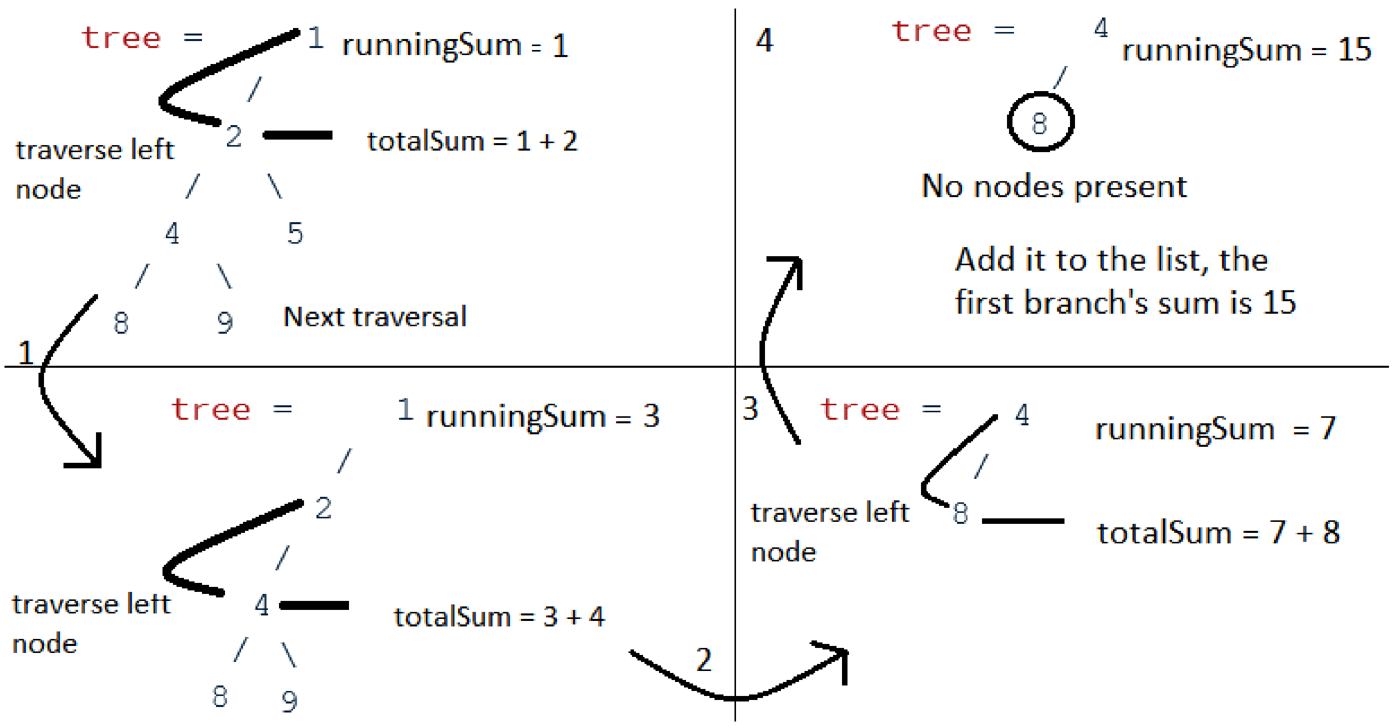
So whenever we reach the end of our tree, i.e, the leaf node, then we will add the total sum to our array.

```
private static void GetBranchSum(BinaryTree tree, int runningSum,
List<int> branchSum)
{
     if (tree == null) return;

     int totalSum = runningSum + tree.value;
     if (tree.left == null && tree.right == null)
     {
```

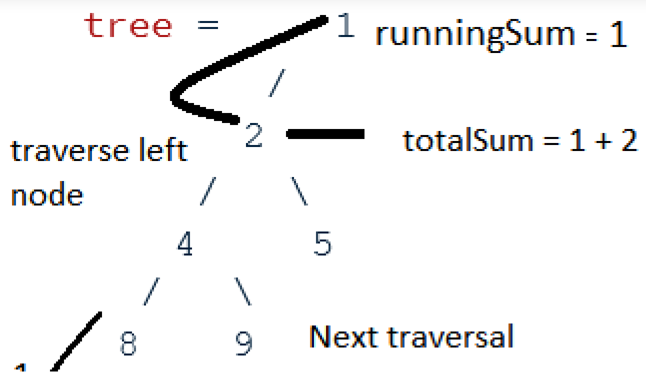
Sign In Get started

But we have to still calculate the sum for our left branch and right branch. Understand it via the below image:

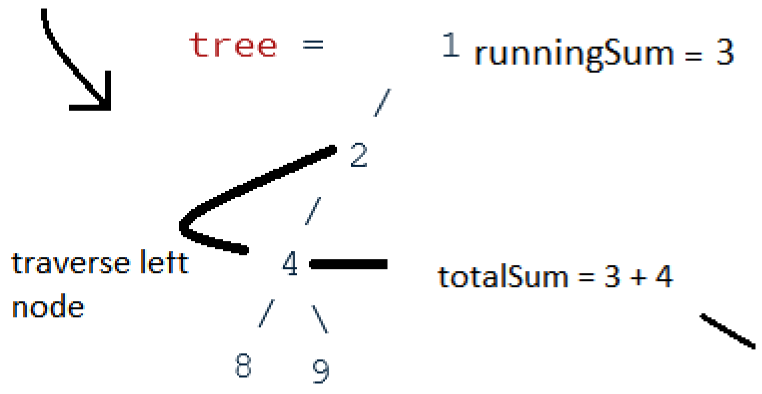


Calculate the first branch's sum

If you didn't understand, then read from here or directly jump to the code part.

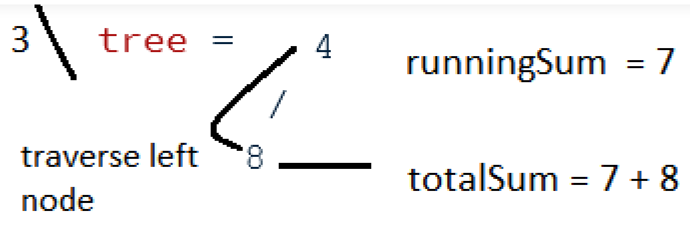


1. *Initially, we were at 1st node, where runningSum = 1, so now we moved to the second node (left branch), so now the totalSum = runningSum + currentNodeValue, thus totalSum = 1 +2 => 3.*

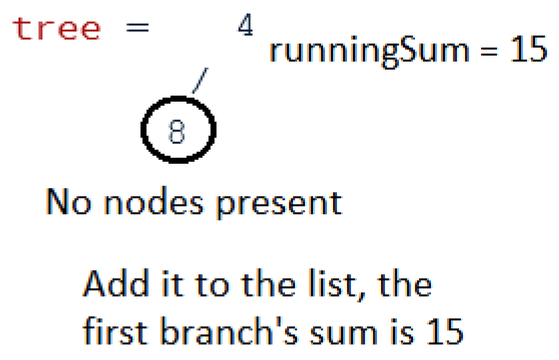


2. *Traversing to next node, again left branch, so now the runningSum = 3, so again,*

Sign In Get started



3. Traversing to next node, again left branch, so now the runningSum = 7, so again, the totalSum = 7+ 8 => 15



But now there's no left node, no right node, this 8 is a leaf node, so that's our base condition, so now we will add the totalSum to the list and will return.

Sign In Get started

```
private static void GetBranchSum(BinaryTree tree, int runningSum,
List<int> branchSum)
{
     if (tree == null) return;

     int totalSum = runningSum + tree.value;
     if (tree.left == null && tree.right == null)
     {
          branchSum.Add(totalSum);
          return;
     }


     GetBranchSum(tree.left, totalSum, branchSum);
     GetBranchSum(tree.right, totalSum, branchSum);
}
```

The **time complexity** for the solution is **O(n)** because we have to traverse each element in the tree and the **space complexity** is also **O(n)** because of the recursion.

Thank you so much for reading till the end. See you in the next article. Till then… **HAPPY LEARNING!!**

Do visit our YouTube channel Code Wrestling for more interesting stuff.