ABHISHEK KUMAR SINGH

Articles Book Notes About Me

Branch Sums

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

JS

Copy

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

Sample Output

JS

Copy

```js
[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
```

Hints

Hint 1

Try traversing the Binary Tree in a depth-first-search-like fashion.

Hint 2

Recursively traverse the Binary Tree in a depth-first-search-like fashion, and pass a running sum of the values of every previously-visited node to each node that you're traversing.

Hint 3

As you recursively traverse the tree, if you reach a leaf node (a node with no "left" or "right" Binary Tree nodes), add the relevant running sum that you've calculated to a list of sums (which you'll also have to pass to the recursive function). If you reach a node that isn't a leaf node, keep recursively traversing its children nodes, passing the correctly updated running sum to them.

Optimal Space & Time Complexity

O(n) time | O(n) space - where n is the number of nodes in the Binary Tree

JS

Copy

```js
// This is the class of the input root.
// Do not edit it.
class BinaryTree {
  constructor(value) {
    this.value = value;
    this.left = null;
    this.right = null;
  }
}

function branchSums(root) {
    const sums = []
  findSumOfNode(root, 0, sums)
    return sums
}

```

```
function findSumOfNode(node, runningSum, sums) {
    // create a array of nodes
    if (!node) return;

    const newRunningSum = runningSum + node.value;
    if(!node.left && !node.right) {
        sums.push(newRunningSum);
        return;
    }

    findSumOfNode(node.left, newRunningSum, sums);
    findSumOfNode(node.right, newRunningSum, sums);

}
```

SHARE ARTICLE Twitter Facebook LinkedIn Mail to Friend

← Node Depth

Do you have any questions, or simply wish to contact me privately? Don't hesitate to shoot me a DM on Twitter.

Have a wonderful day.

Abhishek 🙏

ABHISHEK KUMAR SINGH

Thanks for reading. It makes a difference. I'll try to help developers with tutorials & blogs. Blogs on design, development and happy-productive life. And it will also support and motivate me to share my knowledge with the community.

PAGES

Articles
Uses
Book Notes
Now
About
Favourite Links
SiteMap
Algorithms
RSS

SOCIAL

Twitter
Instagram
LinkedIn