113. Path Sum II

Feb. 1, 2020 | 8.2K views

Average Rating: 4.50 (12 votes)

Given a binary tree and a sum, find all root-to-leaf paths where each path's sum equals the given sum.

Note: A leaf is a node with no children.

Example:

Given the below binary tree and `sum = 22`,

```
      5
     / \
    4   8
   /   / \
  11  13  4
 /  \    / \
7    2  5   1
```

Return:

```
[
   [5,4,11,2],
   [5,8,4,5]
]
```

Solution

Approach: Depth First Traversal | Recursion

Intuition

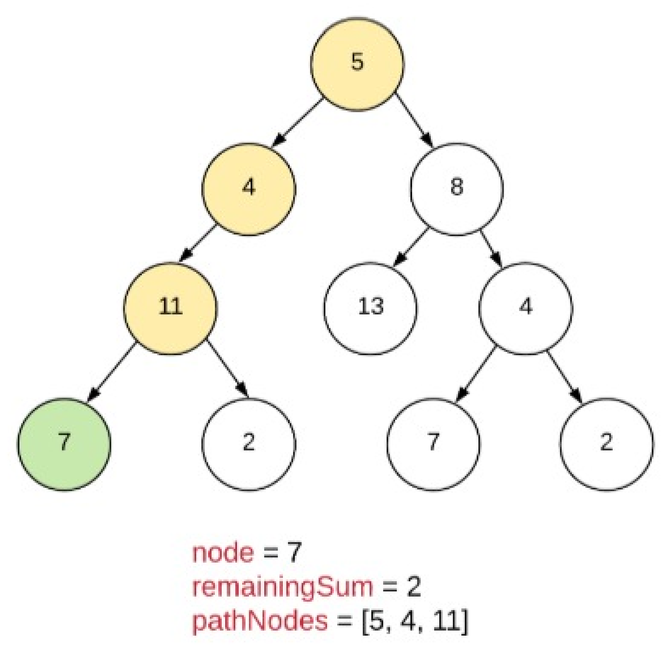
The intuition for this approach is pretty straightforward. The problem statement asks us to find *all* root to leaf paths in a given binary tree. If you simply consider the depth first traversal on a tree, all it does is traverse once branch after another. All we need to do here is to simply execute the depth first traversal and maintain two things along the way:

1. A running sum of all the nodes traversed till that point in recursion and
2. A list of all those nodes

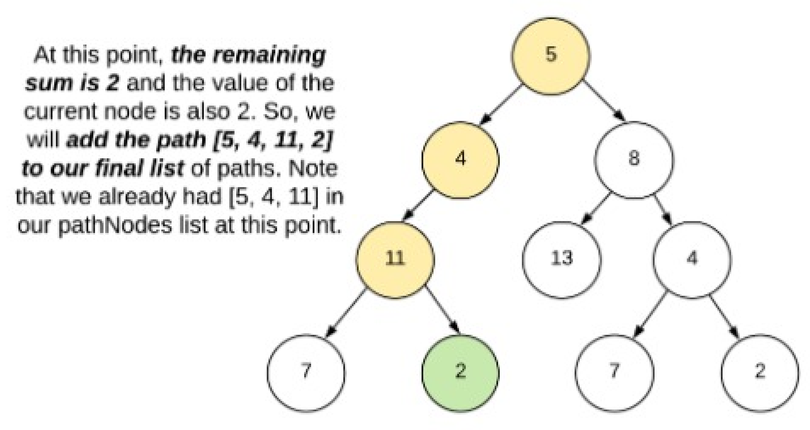
If ever the sum becomes equal to the required sum, and the node where this happens is a leaf node, we can simply add the list of nodes to our final solution. We keep on doing this for every branch of the tree and we will get all the root to leaf paths in this manner that add up to a certain value. Basically, these paths are branches and hence the depth first traversal makes the most sense here. We can also use the breadth first approach for solving this problem. However, that would be super heavy on memory and is not a recommended approach for this very problem. We will look into more details towards the end.

Algorithm

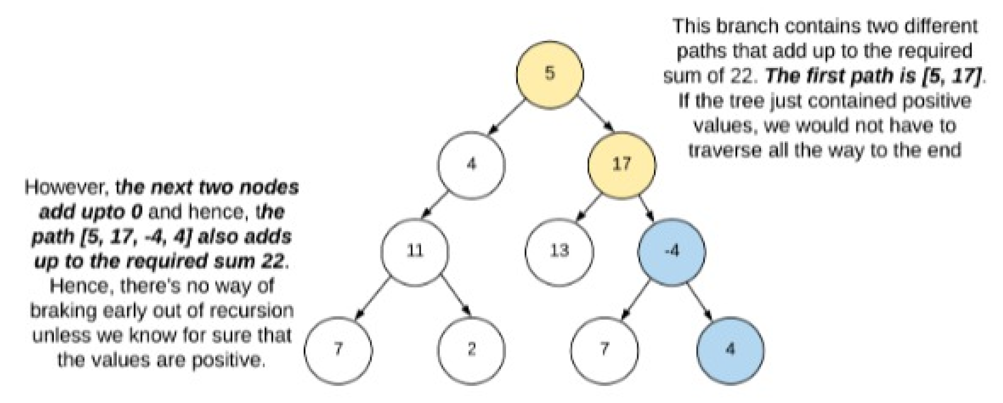
1. We'll define a function called `recurseTree` which will take the following parameters
   - `node` which represents the current node we are on during recursion
   - `remainingSum` which represents the remaining sum that we need to find going down the tree. We can also pass the current sum in our recursive calls. However, then we'd also need to pass the required sum as an additional variable since we'd have to compare the current sum against that value. By passing in remaining sum, we can avoid having to pass additional variable and just see if the remaining sum is 0 (or equal to the value of the current node).
   - Finally, we'll have to pass a list of nodes here which will simply contain the list of all the nodes we have seen till now on the current branch. Let's call this `pathNodes`.
   - The following examples assume the sum to be found is 22.



2. At every step along the way, we will simply check if the remaining sum is equal to the value of the current node. If that is the case and the current node is a leaf node, we will add the current `pathNodes` to the final list of paths that we have to return as a result.



3. The problem statement here specifically mentions `root to leaf` paths. A slight *modification* is to find all `root to node paths`. The solutions are almost similar except that we'd get rid of the *leaf check*.
   - An important thing to consider for this modification is that the problem statement doesn't mention anything about the values of the nodes. That means, we can't assume them to be positive. Had the values been positive, we could stop at the node where the sum became equal to the node's value.
   - However, if the values of the nodes can be negative, then we have to traverse all the branches, all the way up to the roots. Let's look at a modified tree for reference.



4. We process one node at a time and every time the value of the remaining sum becomes equal to the value of the current node, we add the `pathNodes` to our final list. Let's go ahead and look at the implementation for this algorithm.

Java Python Copy

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:

    def recurseTree(self, node, remainingSum, pathNodes, pathsList):

        if not node:
            return

        # Add the current node to the path's list
        pathNodes.append(node.val)

        # Check if the current node is a leaf and also, if it
        # equals our remaining sum. If it does, we add the path to
        # our list of paths
        if remainingSum == node.val and not node.left and not node.right:
            pathsList.append(list(pathNodes))
        else:
            # Else, we will recurse on the left and the right children
            self.recurseTree(node.left, remainingSum - node.val, pathNodes, pathsList)
            self.recurseTree(node.right, remainingSum - node.val, pathNodes, pathsList)

```

Complexity Analysis

- Time Complexity: $O(N^2)$ where N are the number of nodes in a tree. In the worst case, we could have a complete binary tree and if that is the case, then there would be $N/2$ leafs. For every leaf, we perform a potential $O(N)$ operation of copying over the `pathNodes` nodes to a new list to be added to the final `pathsList`. Hence, the complexity in the worst case could be $O(N^2)$.
- Space Complexity: $O(N)$. The space complexity, like many other problems is debatable here. I personally choose *not* to consider the space occupied by the `output` in the space complexity. So, all the `new` lists that we create for the paths are actually a part of the output and hence, don't count towards the final space complexity. The only *additional* space that we use is the `pathNodes` list to keep track of nodes along a branch.

We could include the space occupied by the new lists (and hence the output) in the space complexity and in that case the space would be $O(N^2)$. There's a great answer on Stack Overflow about whether to consider input and output space in the space complexity or not. I prefer *not* to include them.

Why Breadth First Search is bad for this problem?

We did touch briefly on this in the intuition section. BFS would solve this problem perfectly. However, note that the problem statement actually asks us to return a list of all the paths that add up to a particular sum. Breadth first search moves one level at a time. That means, we would have to maintain the `pathNodes` lists for `all` the paths till a particular level/depth at the same time.

Say we are at the level 10 in the tree and that level has e.g. 20 nodes. BFS uses a queue for processing the nodes. Along with 20 nodes in the queue, we would also need to maintain 20 different `pathNodes` lists since there is no backtracking here. That is too much of a space overhead.

The good thing about depth first search is that it uses recursion for processing one branch at a time and once we are done processing the nodes of a particular branch, we `pop` them from the `pathNodes` list thus saving on space. At a time, this list would only contain all the nodes in a single branch of the tree and nothing more. Had the problem statement asked us the total number of paths that add up to a particular sum (root to leaf), then breadth first search would be an equally viable approach.

Rate this article:

Previous Next

Comments: 8 Sort By

Type comment here... (Markdown is supported)

Preview Post

HumanAfterA11 19 February 5, 2020 9:39 AM

Recursion time compexity should be nlogn, the max list length is logn, and n/2 leafs

15 Share Reply

SHOW 8 REPLIES

aman_agarwal2189 9 March 15, 2020 10:26 AM

The complexity should be nlogn.
Every path from root to leaf would be at most logN elements. and there would be atmost n/2 such path (n/2 = number of leaves in a tree).
Am i missing something here?

7 Share Reply

SHOW 1 REPLY

innerpieces 4 May 8, 2020 10:31 AM

The time complexity should be o(n) we are only visiting each node once

2 Share Reply

nishadkumar 97 February 2, 2020 7:36 AM

Nice explanation! If this is solved iteratively, will that be a disadvantage or debatable? I know recursive solution is better with respect to readability. But what is the trade off if the call stack gets bigger than necessary? Thank you though.

2 Share Reply

SHOW 4 REPLIES

prithul 3 June 25, 2020 2:23 AM

hey can anyone explain why is time comoplexity not O(n) here . We are visiting every node only once.

3 Share Reply

SHOW 1 REPLY

bharaniabhishek123 0 May 19, 2020 10:46 PM

there is minor mistake in tree picture above , on the right side, last two leaf nodes should be 5 and 1

0 Share Reply

youning0701 0 May 8, 2020 1:05 PM

Why the following code is not correct?

```
class Solution:
    def pathSum(self, root: TreeNode, sum: int) -> List[List[int]]:
        res=[]
```

Read More

0 Share Reply

SHOW 1 REPLY

leklek0816 2 March 27, 2020 11:59 AM

44 ms, faster than 71.00% of Python3 online submissions for Path Sum II.

```
def pathSum(self, root, target):
    if not root:
        return []
```

Read More

0 Share Reply