112. Path Sum

Nov. 17, 2018 | 46.6K views

Average Rating: 4.53 (19 votes)

Given a binary tree and a sum, determine if the tree has a root-to-leaf path such that adding up all the values along the path 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      1
```

return true, as there exist a root-to-leaf path `5->4->11->2` which sum is 22.

Solution

Binary tree definition

First of all, here is the definition of the `TreeNode` which we would use in the following implementation.

Java | Python — Copy

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

Approach 1: Recursion

The most intuitive way is to use a recursion here. One is going through the tree by considering at each step the node itself and its children. If node *is not* a leaf, one calls recursively `hasPathSum` method for its children with a sum decreased by the current node value. If node *is* a leaf, one checks if the the current sum is zero, *i.e* if the initial sum was discovered.

Java | Python — Copy

```
class Solution:
    def hasPathSum(self, root, sum):
        """
        :type root: TreeNode
        :type sum: int
        :rtype: bool
        """
        if not root:
            return False

        sum -= root.val
        if not root.left and not root.right:  # if reach a leaf
            return sum == 0
        return self.hasPathSum(root.left, sum) or self.hasPathSum(root.right, sum)
```

Complexity Analysis

- Time complexity : we visit each node exactly once, thus the time complexity is $\mathcal{O}(N)$, where N is the number of nodes.
- Space complexity : in the worst case, the tree is completely unbalanced, *e.g.* each node has only one child node, the recursion call would occur N times (the height of the tree), therefore the storage to keep the call stack would be $\mathcal{O}(N)$. But in the best case (the tree is completely balanced), the height of the tree would be $\log(N)$. Therefore, the space complexity in this case would be $\mathcal{O}(\log(N))$.

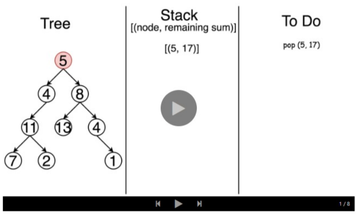
Approach 2: Iterations

Algorithm

We could also convert the above recursion into iteration, with the help of stack. DFS would be better than BFS here since it works faster except the worst case. In the worst case the path `root->leaf` with the given sum is the last considered one and in this case DFS results in the same productivity as BFS.

> The idea is to visit each node with the DFS strategy, while updating the remaining sum to cumulate at each visit.

So we start from a stack which contains the root node and the corresponding remaining sum which is `sum - root.val`. Then we proceed to the iterations: pop the current node out of the stack and return `True` if the remaining sum is `0` and we're on the leaf node. If the remaining sum is not zero or we're not on the leaf yet then we push the child nodes and corresponding remaining sums into stack.



Java | Python — Copy

```
class Solution:
    def hasPathSum(self, root, sum):
        """
        :type root: TreeNode
        :type sum: int
        :rtype: bool
        """
        if not root:
            return False

        de = [(root, sum - root.val), ]
        while de:
            node, curr_sum = de.pop()
            if not node.left and not node.right and curr_sum == 0:
                return True
            if node.right:
                de.append((node.right, curr_sum - node.right.val))
            if node.left:
                de.append((node.left, curr_sum - node.left.val))
        return False
```

Complexity Analysis

- Time complexity : the same as the recursion approach $\mathcal{O}(N)$.
- Space complexity : $\mathcal{O}(N)$ since in the worst case, when the tree is completely unbalanced, *e.g.* each node has only one child node, we would keep all N nodes in the stack. But in the best case (the tree is balanced), the height of the tree would be $\log(N)$. Therefore, the space complexity in this case would be $\mathcal{O}(\log(N))$.

Rate this article:

Comments: 14

Sort By

Type comment here... (Markdown is supported)

Preview Post

hongyu9310 ★ 34 December 28, 2018 12:58 PM

I'm pretty sure this is DFS

node, curr_sum = de.pop()
if node.right: de.append(node.right, curr_sum - node.right.val)
if node.left: de.append(node.left, curr_sum - node.left.val)

Read More

8 Share Reply

Nideesh7 ★ 878 June 24, 2019 10:41 AM

Java Code + Whiteboard Youtube Video Explanation accepted - https://www.youtube.com/watch?v=zfxRQGdMTiE (clickable link)

Read More

6 Share Reply

pidoubi ★ 10 January 6, 2019 2:53 AM

If a stack there then it is dfs
Bfs should use a queue

5 Share Reply

r061206 ★ 735 July 6, 2019 8:28 AM

For iteration version, it can also use pair:

```
public boolean hasPathSum(TreeNode root, int sum) {
    LinkedList<Pair<TreeNode, Integer>> stack = new LinkedList<>();
    stack.add(new Pair(root, sum));
```

Read More

4 Share Reply

LeetMachine ★ 3 April 2, 2020 2:56 AM

@andvary Can this be done using morris traversal in constant space?

1 Share Reply

SHOW 2 REPLIES

yyc0340 ★ 2 November 23, 2018 10:51 AM

Correct me from wrong. The iterative solution seems to be BFS rather than DFS.

0 Share Reply

SHOW 2 REPLIES

silviuh1 ★ 0 2 days ago

```
class Solution(object):
    def hasPathSum(self, root, sum):

        def helper(root, partial_sum):
```

Read More

0 Share Reply

origamous ★ 0 May 25, 2020 10:04 PM

```
class Solution {
    func hasPathSum(_ root: TreeNode?, _ sum: Int) -> Bool {
        guard let root = root else { return false }
```

Read More

0 Share Reply

ramster00 ★ 7 May 24, 2020 7:29 AM

why is [1,2] with target 1 a False, when [1] with a target of 1 is True?

0 Share Reply

SHOW 1 REPLY

srihank ★ 149 May 7, 2020 10:08 AM

The recursive version is straightforward.

```
    public boolean hasPathSum(TreeNode root, int sum) {
        if (root == null) return false;
        sum -= root.val;
```

Read More

0 Share Reply

1 2