666. Path Sum IV

Dec. 10, 2017 | 10.7K views

Average Rating: 3.86 (7 votes)

If the depth of a tree is smaller than `5`, then this tree can be represented by a list of three-digits integers.

For each integer in this list:

1. The hundreds digit represents the depth `D` of this node, `1 <= D <= 4.`
2. The tens digit represents the position `P` of this node in the level it belongs to, `1 <= P <= 8`. The position is the same as that in a full binary tree.
3. The units digit represents the value `V` of this node, `0 <= V <= 9.`

Given a list of `ascending` three-digits integers representing a binary tree with the depth smaller than 5, you need to return the sum of all paths from the root towards the leaves.

Example 1:

```
Input: [113, 215, 221]
Output: 12
Explanation:
The tree that the list represents is:
    3
   / \
  5   1

The path sum is (3 + 5) + (3 + 1) = 12.
```

Example 2:

```
Input: [113, 221]
Output: 4
Explanation:
The tree that the list represents is:
    3
     \
      1

The path sum is (3 + 1) = 4.
```

Approach #1: Convert to Tree [Accepted]

Intuition

Convert the given array into a tree using Node objects. Afterwards, for each path from root to leaf, we can add the sum of that path to our answer.

Algorithm

There are two steps, the tree construction, and the traversal.

In the tree construction, we have some depth, position, and value, and we want to know where the new node goes. With some effort, we can see the relevant condition for whether a node should be left or right is `pos - 1 < 2**(depth - 2)`. For example, when `depth = 4`, the positions are `1, 2, 3, 4, 5, 6, 7, 8`, and it's left when `pos <= 4`.

In the traversal, we perform a depth-first search from root to leaf, keeping track of the current sum along the path we have travelled. Every time we reach a leaf `(node.left == null && node.right == null)`, we have to add that running sum to our answer.

Java | Python — Copy

```
class Node(object):
    def __init__(self, val):
        self.val = val
        self.left = self.right = None

class Solution(object):
    def pathSum(self, nums):
        self.ans = 0
        root = Node(nums[0] % 10)

        for x in nums[1:]:
            depth, pos, val = x/100, x/10 % 10, x % 10
            pos -= 1
            cur = root
            for d in xrange(depth - 2, -1, -1):
                if pos < 2**d:
                    cur.left = cur = cur.left or Node(val)
                else:
                    cur.right = cur = cur.right or Node(val)

                pos %= 2**d

        def dfs(node, running_sum = 0):
            if not node: return
            running_sum += node.val
            if not node.left and not node.right:
                self.ans += running_sum
```

Complexity Analysis

- Time Complexity: $O(N)$ where N is the length of `nums`. We construct the graph and traverse it in this time.
- Space Complexity: $O(N)$, the size of the implicit call stack in our depth-first search.

Approach #2: Direct Traversal [Accepted]

Intuition and Algorithm

As in *Approach #1*, we will depth-first search on the tree. One time-saving idea is that we can use `num / 10 = 10 * depth + pos` as a unique identifier for that node. The left child of such a node would have identifier `10 * (depth + 1) + 2 * pos - 1`, and the right child would be one greater.

Java | Python — Copy

```
class Solution(object):
    def pathSum(self, nums):
        self.ans = 0
        values = {x / 10: x % 10 for x in nums}
        def dfs(node, running_sum = 0):
            if node not in values: return
            running_sum += values[node]
            depth, pos = divmod(node, 10)
            left = (depth + 1) * 10 + 2 * pos - 1
            right = left + 1

            if left not in values and right not in values:
                self.ans += running_sum
            else:
                dfs(left, running_sum)
                dfs(right, running_sum)

        dfs(nums[0] / 10)
        return self.ans
```

Complexity Analysis

- Time and Space Complexity: $O(N)$. The analysis is the same as in *Approach #1*.

Analysis written by: @awice.

Rate this article:

Previous Next

Comments: 5

Sort By

Type comment here... (Markdown is supported)

Preview Post

jinsankim707 ★ 2 June 20, 2018 6:04 AM

cur.left = cur = cur.left or Node(val)

does anyone know what this does?

2 Share Reply

SHOW 3 REPLIES

lenchen1112 ★ 605 December 23, 2019 2:37 PM

A bottom-up approach, O(n)/O(n) time/space.

```
# Python3
import collections
class Solution:
```

Read More

1 Share Reply

charlie11 ★ 129 November 26, 2018 8:21 PM

Pls see my solution which plays around the natural rules on indexes for tree and no DFS needed: https://leetcode.com/problems/path-sum-iv/discuss/198341/Simple-Java-Smart-Solution-(Map-tree-greater-Find-leaves-greater-Calc-Path-for-leaves)

0 Share Reply

i_tyagi ★ 8 June 7, 2019 11:32 AM

```
class Solution {
public:
    void dfs(vector<vector< pair<int, int> > >&g, int l, int &sum, int cur, int i
```

Read More

0 Share Reply

anonX ★ 6 January 3, 2020 2:52 PM

Change the / to // (integer division) in the Python solution.

0 Share Reply