Daily Coding Problem #196

Problem

This problem was asked by Apple.

Given the root of a binary tree, find the most frequent subtree sum. The subtree sum of a node is the sum of all values under a node, including the node itself.

For example, given the following tree:

```
  5
 / \
2  -5
```

Return 2 as it occurs twice: once as the left leaf, and once as the sum of `2 + 5 - 5`.

Solution

A straightforward way to solve this problem would be to keep a dictionary or counter, and then traverse the tree while recursively computing each subtree

sum. We'll do this by computing the leaves first, and then use the solution to that to calculate the subtree sums of its ancestors. At each step we'll increment the counter for that specific sum.

Once we've finished, we can look at the counter and find the key with the highest occurring value -- that's our most frequent subtree sum.

```
from collections import Counter

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

def frequent_subtree_sum(root):
    if root is None:
        return None

    c = Counter()

    def get_subtree_sum(node):
        if node is None:
            return 0
        s = node.val + get_subtree_sum(node.left) + get_subtree_sum(node.right)
        c[s] += 1
        return s

    get_subtree_sum(root)
    return c.most_common(1)[0]
```

This takes O(n) time and space, since we have to traverse the entire tree once and store each subtree sums' counts.