Daily Coding Problem #94

Problem

This problem was asked by Google.

Given a binary tree of integers, find the maximum path sum between two nodes. The path must go through at least one node, and does not need to go through the root.

Solution

We can solve this problem recursively. There are three cases that the max-sum path can fall under:

1. The path includes the root value
2. The path is the max-sum path of the left subtree
3. The path is the max-sum path of the right subtree

Our algorithm will return the maximum of these three values. However, solving for the maximum sum alone in cases 2 and 3 does not solve case 1. In order for the root to join the left subtree's path, a path must end at the root of the subtree. The same applies to the right subtree. Thus, we not only return the maximum sum of the path, but also the height of the tallest path ending at the root of the subtree. The base case is when the node is a single-node tree, which would mean that the maximum sum can only be its own value, and the height is 1.

```
def max_path_sum(self, root):
    def helper(root):
        if root is None:
            return (float('-inf'), 0)

        left_max_sum, left_path = helper(root.left)
        right_max_sum, right_path = helper(root.right)
        # Calculates the maximum path through the root
        root_max_sum = max(0, left_path) + root.val + max(0, right_path)
```

```
        # Find the maximum path, including or excluding the root
        max_sum = max(left_max_sum, root_max_sum, right_max_sum)
        # Find the maximum path including and ending at the root
        root_path = max(left_path, right_path, 0) + root.val

        return (max_sum, root_path)

    # Return only the maximum path
    return helper(root)[0]
```

Since our algorithm is similar to a DFS search on a binary tree, the solution has a time complexity of `O(N)`, and uses up to `O(N)` space on the call stack.