Solution Review: Problem Challenge 2

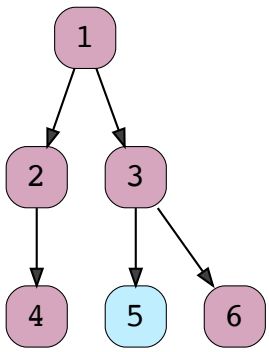
We'll cover the following

Path with Maximum Sum (hard)

Find the path with the maximum sum in a given binary tree. Write a function that returns the maximum sum. A path can be defined as a **sequence of nodes between any two nodes** and doesn't necessarily pass through the root.

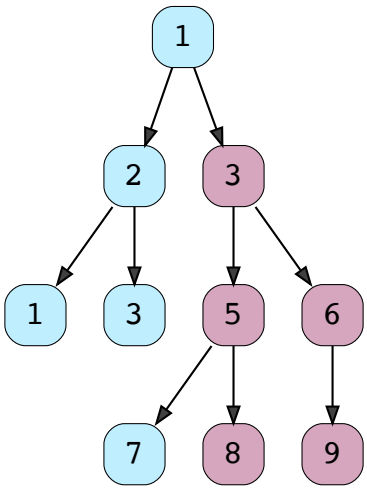
Example 1:

Output: 16
Explanation: The path with maximum sum is: [4, 2, 1, 3, 6]



Example 2:

Output: 31
Explanation: The path with maximum sum is: [8, 5, 3, 6, 9]



Solution

This problem follows the Binary Tree Path Sum (https://www.educative.io/collection/page/5668639101419520/5671464854355968/5642684278505472/) pattern and shares the algorithmic logic with Tree Diameter (https://www.educative.io/collection/page/5668639101419520/5671464854355968/5691878833913856/). We can follow the same **DFS** approach. The only difference will be to ignore the paths with negative sums. Since we need to find the overall maximum sum, we should ignore any path which has an overall negative sum.

Code

Here is what our algorithm will look like, the most important changes are in the highlighted lines:

Java | Python3 | C++ | JS

```
16      return self.globalMaximumSum
17
18    def find_maximum_path_sum_recursive(self, currentNode):
19      if currentNode is None:
20        return 0
21
22      maxPathSumFromLeft = self.find_maximum_path_sum_recursive(
23        currentNode.left)
24      maxPathSumFromRight = self.find_maximum_path_sum_recursive(
25        currentNode.right)
26
27      # ignore paths with negative sums, since we need to find the maximum sum we should
28      # ignore any path which has an overall negative sum.
29      maxPathSumFromLeft = max(maxPathSumFromLeft, 0)
30      maxPathSumFromRight = max(maxPathSumFromRight, 0)
31
32      # maximum path sum at the current node will be equal to the sum from the left subtree
33      # the sum from right subtree + val of current node
34      localMaximumSum = maxPathSumFromLeft + maxPathSumFromRight + currentNode.val
35
36      # update the global maximum sum
37      self.globalMaximumSum = max(self.globalMaximumSum, localMaximumSum)
38
39      # maximum sum of any path from the current node will be equal to the maximum of
40      # the sums from left or right subtrees plus the value of the current node
41      return max(maxPathSumFromLeft, maxPathSumFromRight) + currentNode.val
42
43
```

Time complexity

The time complexity of the above algorithm is $O(N)$, where 'N' is the total number of nodes in the tree. This is due to the fact that we traverse each node once.

Space complexity

The space complexity of the above algorithm will be $O(N)$ in the worst case. This space will be

used to store the recursion stack. The worst case will happen when the given tree is a linked list (i.e., every node has only one child).