Solution Review: Problem Challenge 2

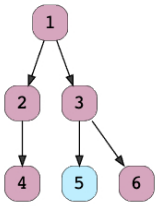
We'll cover the following

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.

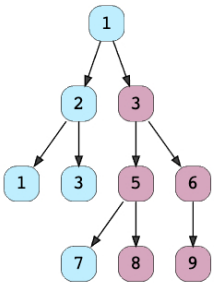
Example 1:

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 pattern and shares the algorithmic logic with Tree Diameter. 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

```
class TreeNode {
  constructor(val, left = null, right = null) {
    this.val = val;
    this.left = left;
    this.right = right;
  }
}


class MaximumPathSum {
  find_maximum_path_sum(root) {
    this.globalMaximumSum = -Infinity;
    this.find_maximum_path_sum_recursive(root);
    return this.globalMaximumSum;
  }

  find_maximum_path_sum_recursive(currentNode) {
    if (currentNode === null) {
      return 0;
    }

    let maxPathSumFromLeft = this.find_maximum_path_sum_recursive(currentNode.left);
    let maxPathSumFromRight = this.find_maximum_path_sum_recursive(currentNode.right);

    // ignore paths with negative sums, since we need to find the maximum sum we should
```

```
26    // ignore any path which has an overall negative sum.
27    maxPathSumFromLeft = Math.max(maxPathSumFromLeft, 0);
28    maxPathSumFromRight = Math.max(maxPathSumFromRight, 0);
```

RUN SAVE RESET

Close

Output 2.413s

```
Maximum Path Sum: 6
Maximum Path Sum: 31
Maximum Path Sum: -1
```

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).