Problem Statement: maximum path sum between any two nodes number binary tree

Asked by Google.
Given a binary tree where each node contains an integer value, find the **maximum path sum** between any two nodes.

- The path must go through at least one node.
- The path **does not** need to pass through the root.
- A path is a sequence of nodes connected by edges, and each node can appear **only once**.

🔍 Example

Given this binary tree:

```
  -10
  /  \
 9   20
     / \
    15  7
```

Output: `42`
Explanation: The path is `15 → 20 → 7` which has a sum of `42`.

Approach

This is a **post-order traversal** problem where we compute:

- For each node, the **maximum path sum including either left or right child**, and
- Track the **maximum path sum including both left and right child plus the current node**.

We'll use a **global variable** to store the result.

Python Code

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

def maxPathSum(root):
```

```
max_sum = float('-inf')  # Global max variable

def helper(node):
    nonlocal max_sum
    if not node:
        return 0

    # Recurse on left and right, ignore negatives
    left = max(helper(node.left), 0)
    right = max(helper(node.right), 0)

    # Current max including both sides
    current = node.val + left + right

    # Update global max
    max_sum = max(max_sum, current)

    # Return max path including only one child
    return node.val + max(left, right)

helper(root)
return max_sum
```

📦 Time and Space Complexity

- **Time Complexity:** `O(N)`, where `N` is the number of nodes (each node visited once).
- **Space Complexity:** `O(H)`, where `H` is the height of the tree (due to recursion stack).