

Maximum Path Sum between 2 Leaf Nodes

Given a binary tree in which each node element contains a number. Find the maximum possible path sum from one leaf node to another leaf node.

Note: Here Leaf node is a node which is connected to exactly one different node.

Example 1:

Input:



Output: 16

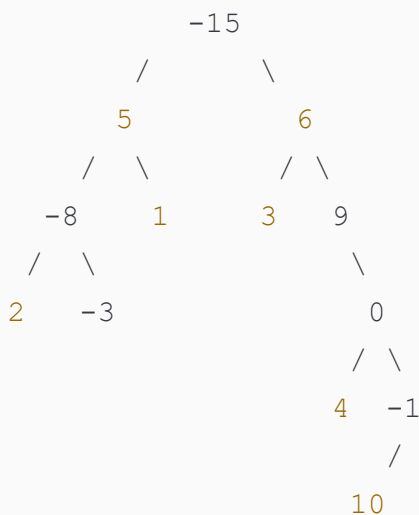
Explanation:

Maximum Sum lies between leaf node 4 and 5.

$4 + 4 + 3 + 5 = 16$.

Example 2:

Input:



Output: 27

Explanation:

The maximum possible sum from one leaf node to another is $(3 + 6 + 9 + 0 + -1 + 10 = 27)$

Your Task:

You don't need to read input or print anything. Complete the function **maxPathSum()** which takes root node as input parameter and returns the maximum sum between 2 leaf nodes.

Expected Time Complexity: $O(N)$

Expected Auxiliary Space: $O(\text{Height of Tree})$

Constraints:

$2 \leq \text{Number of nodes} \leq 10^4$

$-10^3 \leq \text{Value of each node} \leq 10^3$

```
class Solution:
    def maxPathSum(self, root):
        # code here
        if root is None:
            return 0
        maxSum = [-1001]
        self.helper(root, maxSum)
        return maxSum[0]

    def helper(self, root, res):
        if root is None:
            return -1001
        if root.left is None and root.right is None:
            return root.val
        lf = self.helper(root.left, res)
        rt = self.helper(root.right, res)

        if root.left != None and root.right != None:
            res[0] = max(res[0], lf + root.val + rt)
            return max(lf, rt) + root.val
        if root.left is None:
            return rt + root.val
        if root.right is None:
            return lf + root.val
```