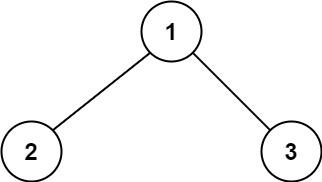
# 题目

路径被定义为一条从树中任意节点出发，沿父节点-子节点连接，达到任意节点的序列。同一个节点在一条路径序列中至多出现一次 。该路径至少包含一个节点，且不一定经过根节点。

路径和是路径中各节点值的总和。

给你一个二叉树的根节点 root ，返回其最大路径和。

示例 1：

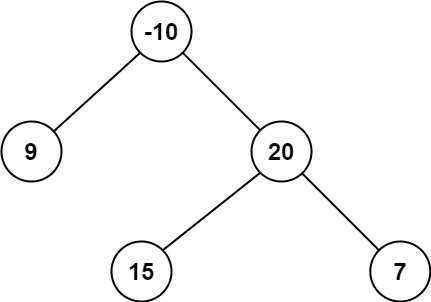


输入：root = [1,2,3]

输出：6

解释：最优路径是 2 -> 1 -> 3 ，路径和为 2 + 1 + 3 = 6

示例 2：



输入：root = [-10,9,20,null,null,15,7]

输出：42

解释：最优路径是 15 -> 20 -> 7 ，路径和为 15 + 20 + 7 = 42

提示：

树中节点数目范围是 [1, 3 \* 10^4]

-1000 <= Node.val <= 1000

# 分析

## 方法一：深度优先遍历

这个问题可以使用递归来解决。对于每个节点，我们可以计算经过该节点的路径和的最大值，同时更新全局的最大路径和。

具体步骤如下：

1、定义一个全局变量maxSum来存储最大路径和的值，初始化为 INT\_MIN。

2、定义一个递归函数maxPathDown(TreeNode\* node)，计算经过节点node的路径和的最大值，并返回该值。

- 如果节点为空，返回 0。

- 递归计算左子树和右子树经过根节点的路径和的最大值leftMax和rightMax。

- 如果leftMax或rightMax为负值，则对应子树的路径和对整体路径和没有贡献，因此取 0。

- 更新maxSum，将当前节点值、leftMax、rightMax三者之和与maxSum比较，取较大值作为新的maxSum。

- 返回经过节点node的路径和的最大值，即node->val + max(leftMax, rightMax)。

3、调用maxPathDown(root)，即可得到最终的最大路径和。

以下是实现这个算法的代码：

class Solution {

public:

int maxSum = INT\_MIN;

int maxPathSum(TreeNode\* root) {

maxPathDown(root);

return maxSum;

}

int maxPathDown(TreeNode\* node) {

if (!node) return 0;

int leftMax = max(0, maxPathDown(node->left));

int rightMax = max(0, maxPathDown(node->right));

maxSum = max(maxSum, node->val + leftMax + rightMax);

return node->val + max(leftMax, rightMax);

}

};

这个算法的时间复杂度是 O(n)，其中 n 是二叉树的节点数。

或：

class Solution {

public:

int maxPathSum(TreeNode\* root) {

ret = INT\_MIN;

dfs(root);

return ret;

}

private:

int ret;

int dfs(TreeNode\* root) {

if (root == nullptr) {

return 0;

}

int left = max(0, dfs(root->left));

int right = max(0, dfs(root->right));

ret = max(ret, left + right + root->val);

return root->val + max(left, right);

}

};