0124. 二叉树中的最大路径和

ITCharge 大约 4 分钟

- 标签：树、深度优先搜索、动态规划、二叉树
- 难度：困难

题目链接

- 0124. 二叉树中的最大路径和 - 力扣

题目大意

描述：给定一个二叉树的根节点 $root$。

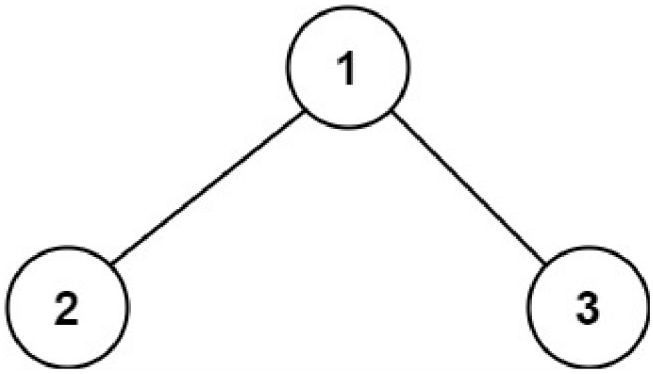
要求：返回其最大路径和。

说明：

- **路径**：被定义为一条节点序列，序 l每对相邻节点之间都存在一条边。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**：路径中各节点值的总和。
- 树中节点数目范围是 $[1, 3 * 10^4]$。
- $-1000 \leq Node.val \leq 1000$。

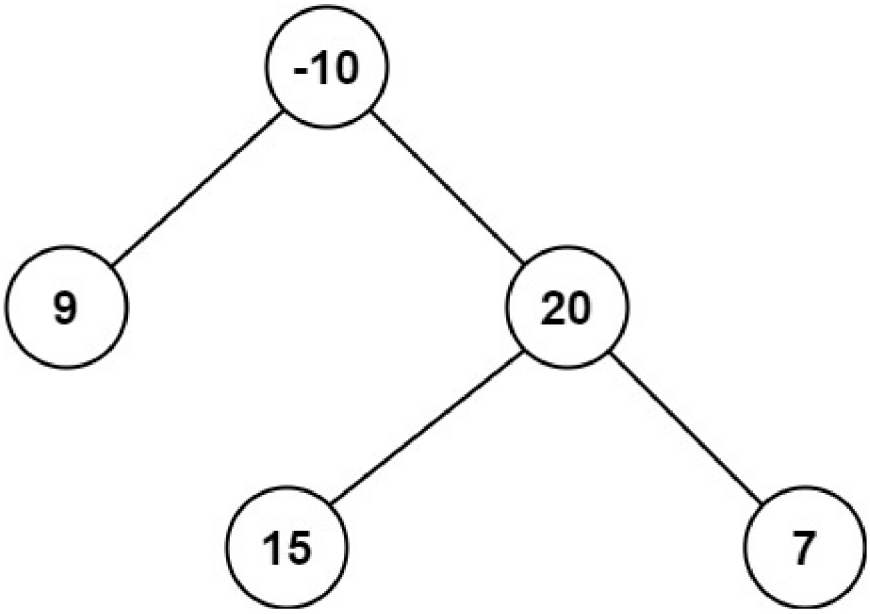
示例：

- 示例 1：



```py
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：




```py
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
```

解题思路

思路 1：树形 DP + 深度优先搜索

根据最大路径和中对应路径是否穿过根节点，我们可以将二叉树分为两种：

1. 最大路径和中对应路径穿过根节点。
2. 最大路径和中对应路径不穿过根节点。

如果最大路径和中对应路径穿过根节点，则：**该二叉树的最大路径和 = 左子树中最大贡献值 + 右子树中最大贡献值 + 当前节点值。**

而如果最大路径和中对应路径不穿过根节点，则：**该二叉树的最大路径和 = 所有子树中最大路径和。**

即：**该二叉树的最大路径和 = max(左子树中最大贡献值 + 右子树中最大贡献值 + 当前节点值，所有子树中最大路径和)**。

对此我们可以使用深度优先搜索递归遍历二叉树，并在递归遍历的同时，维护一个最大路径和变量 ans。

然后定义函数 `def dfs(self, node)`：计算二叉树中以该节点为根节点，并且经过该节点的最大贡献值。

计算的结果可能的情况有 2 种：

1. 经过空节点的最大贡献值等于 0。
2. 经过非空节点的最大贡献值等于 **当前节点值 + 左右子节点提供的最大贡献值中较大的一个**。如果该贡献值为负数，可以考虑舍弃，即最大贡献值为 0。

在递归时，我们先计算左右子节点的最大贡献值，再更新维护当前最大路径和变量。最终 ans 即为答案。具体步骤如下：

1. 如果根节点 $root$ 为空，则返回 0。
2. 递归计算左子树的最大贡献值为 $left\_max$。
3. 递归计算右子树的最大贡献值为 $right\_max$。
4. 更新维护最大路径和变量，即 $self.ans = max\{self.ans, \quad left\_max + right\_max + node.val\}$。
5. 返回以当前节点为根节点，并且经　　节点的最大贡献值。即返回 **当前节点值 + 左右子节点提供的最大贡献值中较大的一个**。
6. 最终 $self.ans$ 即为答案。

思路 1：代码

py

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def __init__(self):
        self.ans = float('-inf')

    def dfs(self, node):
        if not node:
            return 0
```

```
        left_max = max(self.dfs(node.left), 0)      # 左子树提供的最大贡献值
        right_max = max(self.dfs(node.right), 0)    # 右子树提供的最大贡献值

        cur_max = left_max + right_max + node.val  # 包含当前节点和左右子树的最大路径和
        self.ans = max(self.ans, cur_max)          # 更新所有路径中的最大路径和

        return max(left_max, right_max) + node.val # 返回包含当前节点的子树的最大贡献值

    def maxPathSum(self, root: Optional[TreeNode]) -> int:
        self.dfs(root)
        return self.ans
```

思路 1：复杂度分析

- **时间复杂度**：$O(n)$，其中 n 是二叉树的节点数目。
- **空间复杂度**：$O(n)$。递归函数需要用到栈空间，栈空间取决于递归深度，最坏情况下递归深度为 n，所以空间复杂度为 $O(n)$。