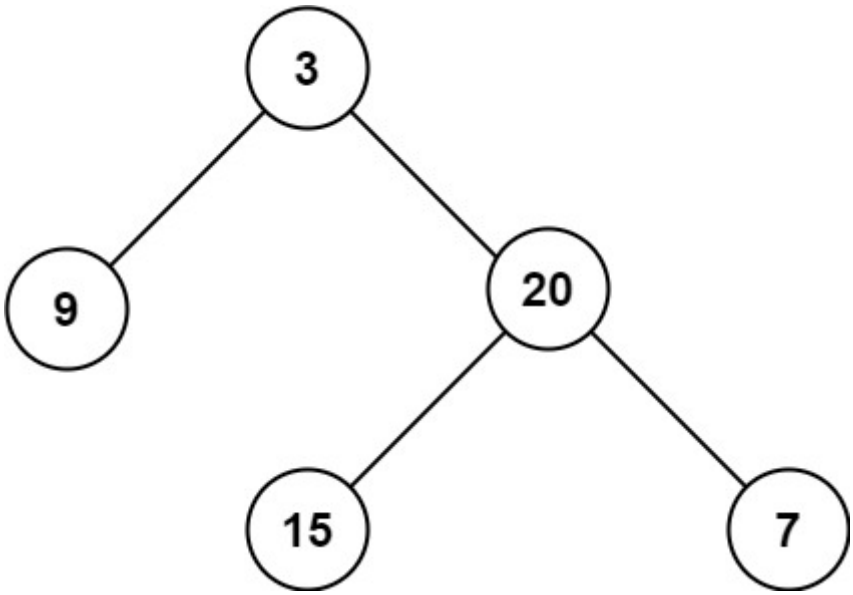
递归

104. 二叉树的最大深度

给定一个二叉树 `root` ，返回其最大深度。

二叉树的 **最大深度** 是指从根节点到最远叶子节点的最长路径上的节点数。

示例 1：




```
输入：root = [3,9,20,null,null,15,7]
输出：3
```

示例 2：

```
输入：root = [1,null,2]
输出：2
```

提示：

- 树中节点的数量在 `[0, 104]` 区间内。
- `-100 <= Node.val <= 100`

方法一：不用全局变量

1. Java

```java
class Solution {
    public int maxDepth(TreeNode root) {
        if (root == null) return 0;
        int lDepth = maxDepth(root.left);
        int rDepth = maxDepth(root.right);
        return Math.max(lDepth, rDepth) + 1;
    }
}
```

2. Python

```python
class Solution:
    def maxDepth(self, root: Optional[TreeNode]) -> int:
        if root is None: return 0
        l_depth = self.maxDepth(root.left)
        r_depth = self.maxDepth(root.right)
        return max(l_depth, r_depth) + 1
```

方法二：用全局变量

1. Java

```java
class Solution {
    private int ans;

    public int maxDepth(TreeNode root) {
        dfs(root, 0);
        return ans;
    }

    private void dfs(TreeNode node, int cnt) {
        if (node == null) return;
        ++cnt;
        ans = Math.max(ans, cnt);
        dfs(node.left, cnt);
        dfs(node.right, cnt);
    }
}
```

2. Python

```python
class Solution:
    def maxDepth(self, root: Optional[TreeNode]) -> int:
        ans = 0
        def dfs(node, cnt):
            if node is None:
                return
            cnt += 1
            nonlocal ans
            ans = max(ans, cnt)
            dfs(node.left, cnt)
            dfs(node.right, cnt)
        dfs(root, 0)
        return ans
```

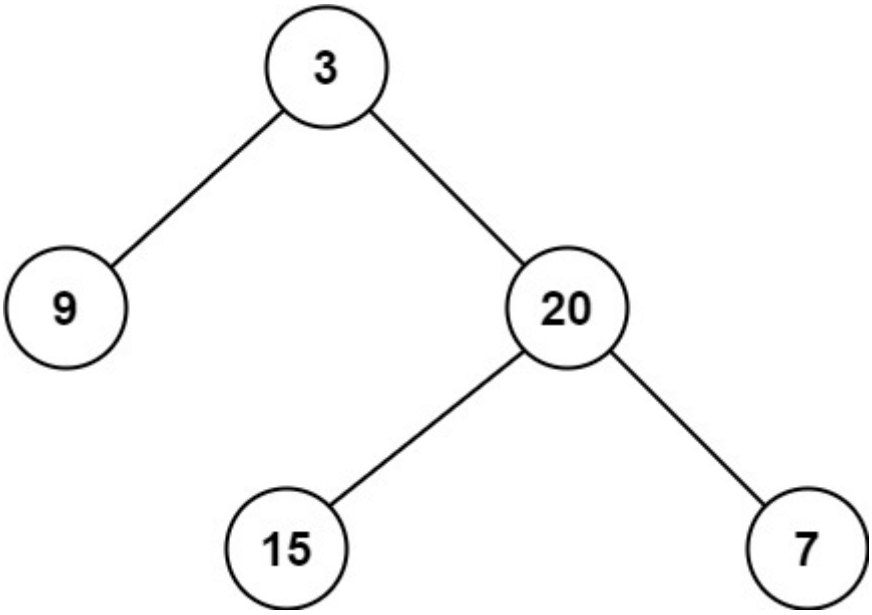
111. 二叉树的最小深度

给定一个二叉树，找出其最小深度。

最小深度是从根节点到最近叶子节点的最短路径上的节点数量。

说明：叶子节点是指没有子节点的节点。

示例 1：




```
输入：root = [3,9,20,null,null,15,7]
输出：2
```

示例 2：

```
输入：root = [2,null,3,null,4,null,5,null,6]
输出：5
```

提示：

- 树中节点数的范围在 `[0, 105]` 内
- `-1000 <= Node.val <= 1000`

注意：

这里不能和求最大深度一样，如果二叉树只有左子树或者右子树，那么按照求最大深度逻辑

$$min(self.minDepth(root.left),\ self.minDepth(root.right)) = 1$$

这显然不符合题意，于是需要对左子树或者右子树不存在的情况进行特判。还有个小优化：递的时候可以最优性剪枝。

方法一：不适用全局变量

1. Java

```java
class Solution {
    public int minDepth(TreeNode root) {
        if (root == null) {
            return 0;
        }
        if (root.right == null) {
            return minDepth(root.left) + 1;
        }
        if (root.left == null) {
            return minDepth(root.right) + 1;
        }
        return Math.min(minDepth(root.left), minDepth(root.right)) + 1;
    }
}
```

2. Python

```python
class Solution:
    def minDepth(self, root: Optional[TreeNode]) -> int:
        if root is None:
            return 0
        if root.right is None:
            return self.minDepth(root.left) + 1
        if root.left is None:
            return self.minDepth(root.right) + 1
        return min(self.minDepth(root.left), self.minDepth(root.right)) + 1
```

方法二：使用全局变量

1. Java

```java
class Solution {
    private int ans = Integer.MAX_VALUE;

    public int minDepth(TreeNode root) {
        dfs(root, 0);
        return root != null ? ans : 0;
    }

    private void dfs(TreeNode node, int cnt) {
        if (node == null || ++cnt >= ans) {
            return;
        }
        if (node.left == node.right) { // node 是叶子
            ans = cnt;
            return;
        }
        dfs(node.left, cnt);
        dfs(node.right, cnt);
    }
}
```

2. Python

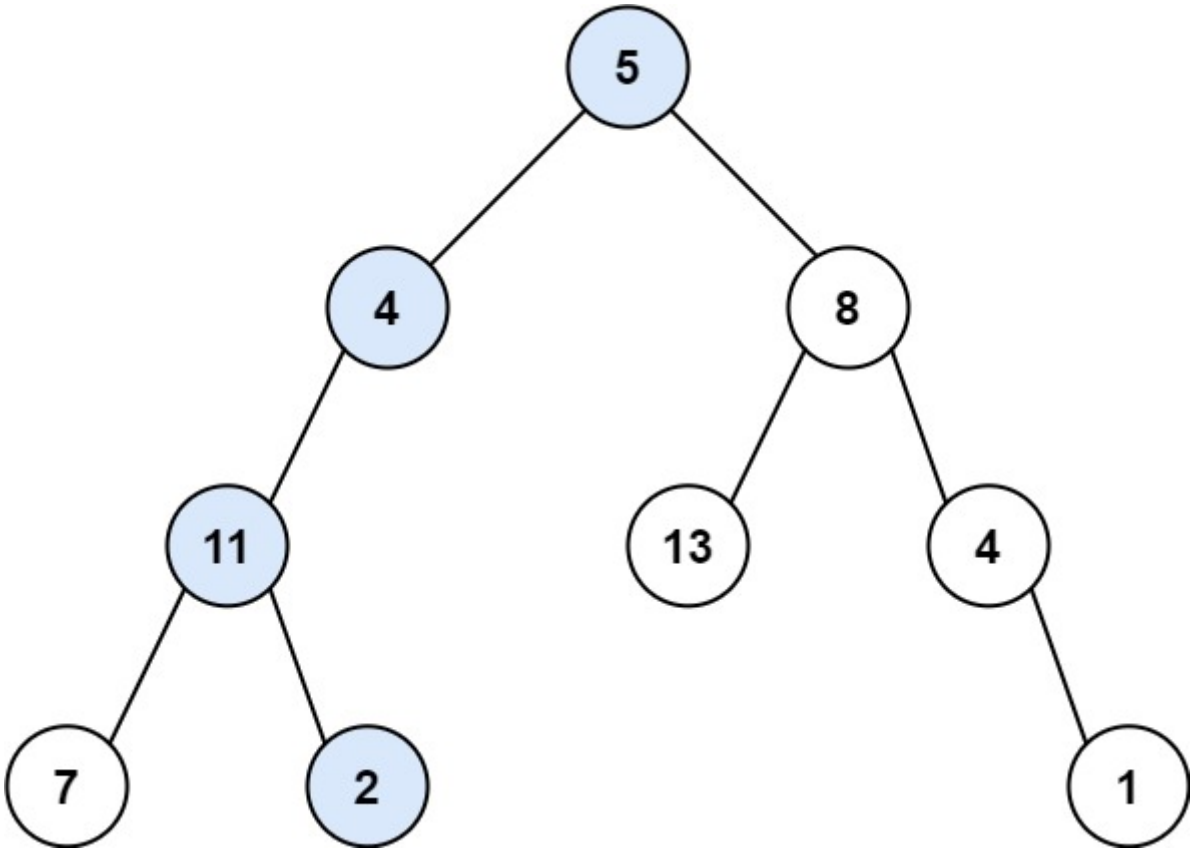
```python
class Solution:
    def minDepth(self, root: Optional[TreeNode]) -> int:
        ans = inf
        def dfs(node: Optional[TreeNode], cnt: int) -> None:
            if node is None:
                return
            nonlocal ans
            cnt += 1
            if cnt >= ans:
                return  # 最优性剪枝
            if node.left is node.right:  # node 是叶子
                ans = cnt
                return
            dfs(node.left, cnt)
            dfs(node.right, cnt)
        dfs(root, 0)
        return ans if root else 0
```

112. 路径总和

给你二叉树的根节点 `root` 和一个表示目标和的整数 `targetSum` 。判断该树中是否存在 **根节点到叶子节点** 的路径，这条路径上所有节点值相加等于目标和 `targetSum` 。如果存在，返回 `true` ；否则，返回 `false` 。

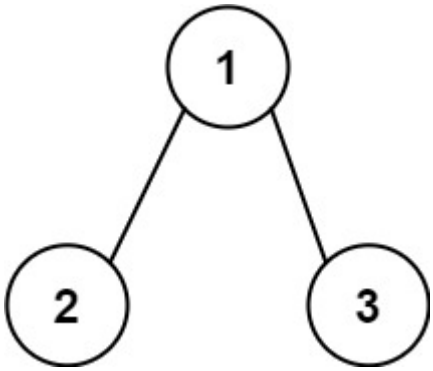
叶子节点 是指没有子节点的节点。

示例 1：



```
输入：root = [5,4,8,11,null,13,4,7,2,null,null,null,1], targetSum = 22
输出：true
解释：等于目标和的根节点到叶节点路径如上图所示。
```

示例 2：




```
输入：root = [1,2,3], targetSum = 5
输出：false
解释：树中存在两条根节点到叶子节点的路径：
(1 --> 2)：和为 3
(1 --> 3)：和为 4
不存在 sum = 5 的根节点到叶子节点的路径。
```

示例 3：

```
输入：root = [], targetSum = 0
输出：false
解释：由于树是空的，所以不存在根节点到叶子节点的路径。
```

提示：

- 树中节点的数目在范围 `[0, 5000]` 内
- `-1000 <= Node.val <= 1000`
- `-1000 <= targetSum <= 1000`

思路：

这题方法比较多，下面给出一种比较简单的思路。

1. 如果 $root$ 是空，直接返回 $false$ 即可。
2. 把 $targetSum$ 减少 $root.val$。
3. 如果 $root$ 是叶子节点，判断 $targetSum == 0$ 即可。
4. 递归左子树或者右子树，只要有一个结果为 $true$ 即可。

1. Java

```java
class Solution {
    public boolean hasPathSum(TreeNode root, int targetSum) {
        if (root == null) {
            return false;
        }
        targetSum -= root.val;
        if (root.left == root.right) { // root 是叶子
            return targetSum == 0;
        }
        return hasPathSum(root.left, targetSum) || hasPathSum(root.right, targetSum);
    }
}
```

2. Python

```python
class Solution:
    def hasPathSum(self, root: Optional[TreeNode], targetSum: int) -> bool:
        if root is None:
            return False
        targetSum -= root.val
        if root.left is root.right:  # root 是叶子
            return targetSum == 0
        return self.hasPathSum(root.left, targetSum) or self.hasPathSum(root.right,
targetSum)
```

129. 求根节点到叶节点数字之和

给你一个二叉树的根节点 `root` ，树中每个节点都存放有一个 `0` 到 `9` 之间的数字。

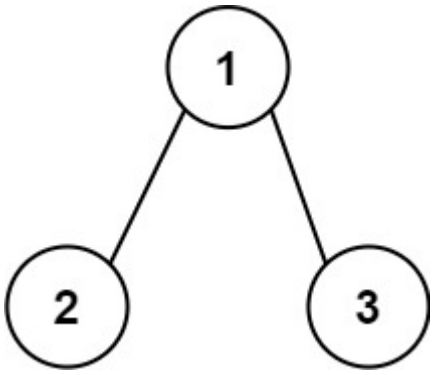
每条从根节点到叶节点的路径都代表一个数字：

- 例如，从根节点到叶节点的路径 `1 -> 2 -> 3` 表示数字 `123` 。

计算从根节点到叶节点生成的 **所有数字之和** 。

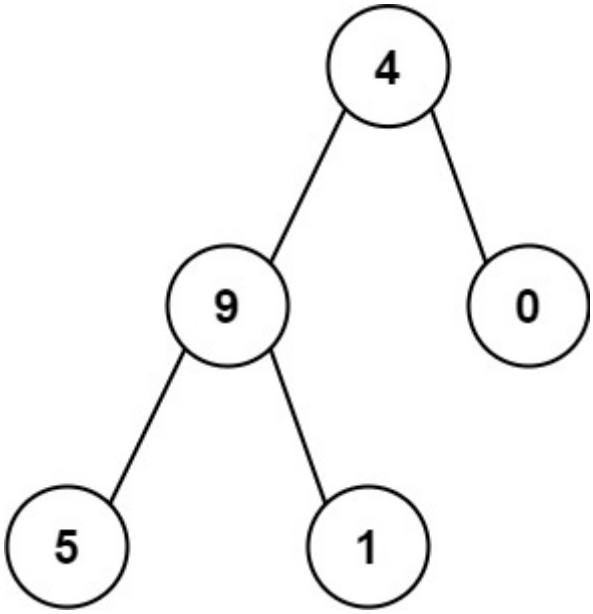
叶节点 是指没有子节点的节点。

示例 1：



```
输入：root = [1,2,3]
输出：25
解释：
从根到叶子节点路径 1->2 代表数字 12
从根到叶子节点路径 1->3 代表数字 13
因此，数字总和 = 12 + 13 = 25
```

示例 2：

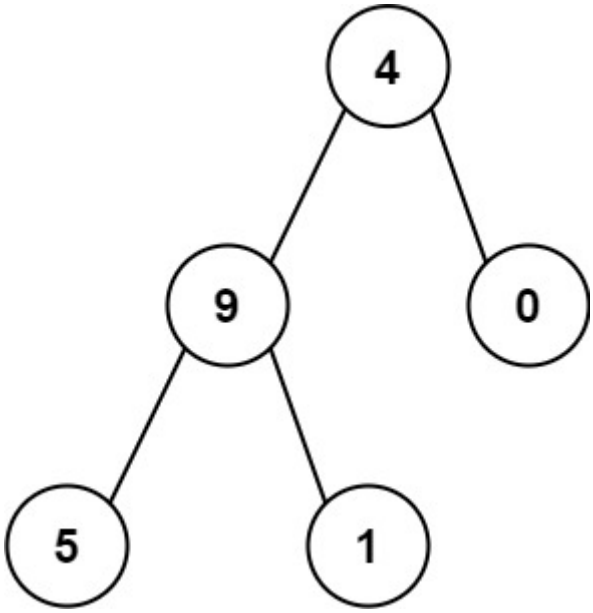



```
输入：root = [4,9,0,5,1]
输出：1026
解释：
从根到叶子节点路径 4->9->5 代表数字 495
从根到叶子节点路径 4->9->1 代表数字 491
从根到叶子节点路径 4->0 代表数字 40
因此，数字总和 = 495 + 491 + 40 = 1026
```

提示：

- 树中节点的数目在范围 `[1, 1000]` 内
- `0 <= Node.val <= 9`
- 树的深度不超过 `10`

思路：



对于路径 $4->9->5$，我们推导一下生成方式。

1. 初始化 $x=0$。
2. 从 4 开始递归，更新 $x=x\cdot 10+4=4$。
3. 向下递归到 9，更新 $x=x\cdot 10+9=49$。
4. 向下递归到 5，更新 $x=x\cdot 10+5=495$。

当我们递归到叶子节点时，把 x 加入到答案中。

算法实现：$x=x\cdot 10+node.val$，x 和 $node$ 作为 dfs 递归的参数。

1. Java

```java
class Solution {
    private int ans;

    private void dfs(TreeNode node, int x) {
        if (node == null) {
            return;
        }
        x = x * 10 + node.val;
        if (node.left == node.right) { // node 是叶子节点
            ans += x;
            return;
        }
        dfs(node.left, x);
        dfs(node.right, x);
    }

    public int sumNumbers(TreeNode root) {
        dfs(root, 0);
        return ans;
    }
}
```

2. Python

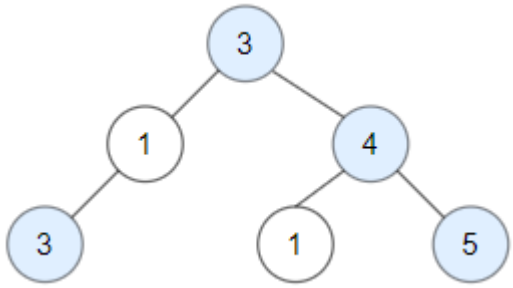
```python
class Solution:
    def sumNumbers(self, root: Optional[TreeNode]) -> int:
        ans = 0
        def dfs(node: Optional[TreeNode], x: int) -> None:
            if node is None:
                return
            x = x * 10 + node.val
            if node.left is node.right:  # node 是叶子节点
                nonlocal ans
                ans += x
                return
            dfs(node.left, x)
            dfs(node.right, x)
        dfs(root, 0)
        return ans
```

1448. 统计二叉树中好节点的数目

给你一棵根为 `root` 的二叉树，请你返回二叉树中好节点的数目。

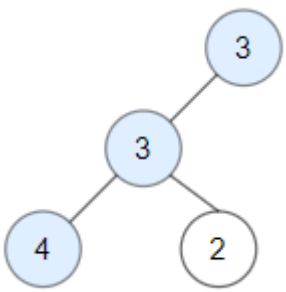
「好节点」X 定义为：从根到该节点 X 所经过的节点中，没有任何节点的值大于 X 的值。

示例 1：




```
输入：root = [3,1,4,3,null,1,5]
输出：4
解释：图中蓝色节点为好节点。
根节点 (3) 永远是个好节点。
节点 4 -> (3,4) 是路径中的最大值。
节点 5 -> (3,4,5) 是路径中的最大值。
节点 3 -> (3,1,3) 是路径中的最大值。
```

示例 2：



```
输入：root = [3,3,null,4,2]
输出：3
解释：节点 2 -> (3, 3, 2) 不是好节点，因为 "3" 比它大。
```

示例 3：

```
输入：root = [1]
输出：1
解释：根节点是好节点。
```

提示：

- 二叉树中节点数目范围是 `[1, 10^5]` 。
- 每个节点权值的范围是 `[-10^4, 10^4]` 。

思路：

用一个变量 mx 来维护从根节点到当前节点的节点最大值即可。

1. 如果 $root$ 为空，直接 $return$ 即可。
2. 途中如果发现 $root.val \geq x$，这里需要更新答案 ans 并且更新 $mx = root.val$。
3. 最后递归根节点和 $-inf$ 即可。

1. Java

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    private int ans = 0;

    public int goodNodes(TreeNode root) {
        dfs(root, Integer.MIN_VALUE);
        return ans;
    }

    private void dfs(TreeNode node, int mx) {
        if (node == null) {
            return;
        }
        if (node.val >= mx) {
            ans++;
```

```
            mx = node.val;
        }
        dfs(node.left, mx);
        dfs(node.right, mx);
    }
}
```

2. Python

```
# 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 goodNodes(self, root: TreeNode) -> int:
        ans = 0
        def f(node, mx):
            if node is None:
                return
            if node.val >= mx:
                nonlocal ans
                ans += 1
                mx = node.val
            f(node.left, mx)
            f(node.right, mx)
        f(root, -inf)
        return ans
```