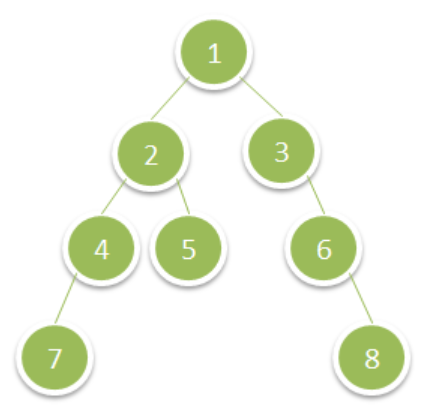
# 描述

给你一棵二叉树，请你返回层数最深的叶子节点的和。

示例 1：



输入：root = [1,2,3,4,5,null,6,7,null,null,null,null,8]

输出：15

示例 2：

输入：root = [6,7,8,2,7,1,3,9,null,1,4,null,null,null,5]

输出：19

**提示：**

树中节点数目在1到10^4之间。

每个节点的值在1到100之间。

# 分析

## 方法一：深度优先搜索

我们可以使用深度优先搜索的方法解决这个问题。

我们从根节点开始进行搜索，在搜索的同时记录当前节点的深度 dep。我们维护两个全局变量 maxdep 和total，其中 maxdep 表示搜索到的节点的最大深度，total 表示搜索到的深度等于 maxdep 的节点的权值之和。

当我们搜索到一个新的节点 x 时，会有以下三种情况：

节点x的深度dep小于maxdep，那么我们可以忽略节点x，继续进行搜索；

节点x的深度dep等于maxdep，那么我们将节点x的权值添加到total中；

节点x的深度dep大于maxdep，此时我们找到了一个深度更大的节点，因此需要将maxdep置为dep，并将total置为节点x的权值。

在深度优先搜索结束之后，深度最大的叶子节点的权值之和即存储在total 中。

代码：

class Solution {

private:

int maxLevel = -1;

int sum = 0;

public:

void dfs(TreeNode\* node, int level) {

if (node == nullptr) {

return;

}

if (level > maxLevel) {

maxLevel = level;

sum = node->val;

} else if (level == maxLevel) {

sum += node->val;

}

dfs(node->left, level + 1);

dfs(node->right, level + 1);

}

int deepestLeavesSum(TreeNode\* root) {

dfs(root, 0);

return sum;

}

};

复杂度分析：

时间复杂度：O(n)，其中n是二叉树的节点数。深度优先搜索需要遍历每个节点一次。

空间复杂度：O(n)，其中n是二叉树的节点数。空间复杂度主要取决于递归调用栈的深度，为二叉树的深度，最坏情况下二叉树的深度是O(n)。

## 方法二：广度优先搜索

我们同样可以使用广度优先搜索的方法解决这个问题。

除了搜索的顺序不同之外，实现的细节与深度优先搜索的方法完全相同。

思路：

要使用广度优先搜索（BFS）来计算二叉树中最深层叶子节点的和，你可以在遍历每一层节点时累加该层的节点值，最终得到最深层节点的和。

代码：

class Solution {

public:

int deepestLeavesSum(TreeNode\* root) {

if (!root) return 0;

queue<TreeNode\*> q;

q.push(root);

int sum = 0;

while (!q.empty()) {

int size = q.size();

sum = 0; // 每层开始前清零

for (int i = 0; i < size; i++) {

TreeNode\* node = q.front();

q.pop();

sum += node->val;

if (node->left) q.push(node->left);

if (node->right) q.push(node->right);

}

}

return sum;

}

};

这个算法使用了队列来实现广度优先搜索。我们首先将根节点入队，然后在每一层的循环中，将该层的所有节点出队，并将它们的值累加到 sum 中。同时，将下一层的节点入队。最后返回 sum 即可。

这个算法的时间复杂度是 O(n)，其中 n 是节点数。