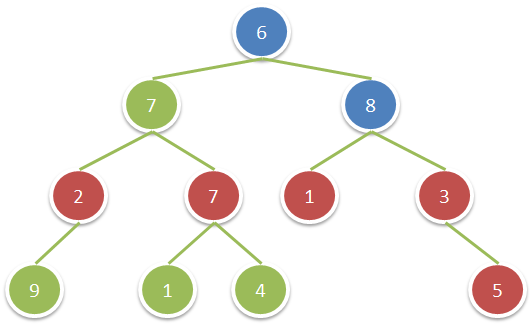
# 描述

给你一棵二叉树，请你返回满足以下条件的所有节点的值之和：

该节点的祖父节点的值为偶数。（一个节点的祖父节点是指该节点的父节点的父节点。）

如果不存在祖父节点值为偶数的节点，那么返回 0 。

**示例：**



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

输出：18

解释：图中红色节点的祖父节点的值为偶数，蓝色节点为这些红色节点的祖父节点。

**提示：**

树中节点的数目在 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

思路：

我们可以通过深度优先搜索找出所有满足题目要求的节点。

具体地，在进行搜索时，搜索状态除了当前节点之外，还需要存储该节点的祖父节点和父节点，即三元组 (grandparent, parent, node)。如果节点 grandparent 的值为偶数，那么就将节点 node 的值加入答案。在这之后，我们继续搜索节点 node 的左孩子 (parent, node, node.left) 以及右孩子 (parent, node, node.right)，直到搜索结束。

代码：

class Solution {

private:

int ans = 0;

public:

void dfs(TreeNode\* grandparent, TreeNode\* parent, TreeNode\* node) {

if (!node) {

return;

}

if (grandparent->val % 2 == 0) {

ans += node->val;

}

dfs(parent, node, node->left);

dfs(parent, node, node->right);

}

int sumEvenGrandparent(TreeNode\* root) {

if (root->left) {

dfs(root, root->left, root->left->left);

dfs(root, root->left, root->left->right);

}

if (root->right) {

dfs(root, root->right, root->right->left);

dfs(root, root->right, root->right->right);

}

return ans;

}

};

复杂度分析：

时间复杂度：O(N)，其中N是树中的节点个数。

空间复杂度：O(H)，其中H是树的高度。

## 方法二：广度优先搜索

我们也可以换一种思考方式。既然要找出祖父节点的值为偶数的节点，我们不如找到所有值为偶数的节点，并对这些节点的孙子节点（即子节点的子节点）统计答案。

这样我们就可以使用广度优先搜索遍历整棵树，当我们找到一个值为偶数的节点时，我们将该节点的所有孙子节点的值加入答案。