# 题目

给定二叉树，按垂序遍历返回其结点值。

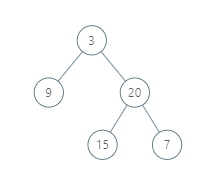
对位于(X, Y)的每个结点而言，其左右子结点分别位于(X-1, Y-1)和(X+1, Y-1)。

把一条垂线从X = -infinity移动到X = +infinity，每当该垂线与结点接触时，我们按从上到下的顺序报告结点的值（Y坐标递减）。

如果两个结点位置相同，则首先报告的结点值较小。

按X坐标顺序返回非空报告的列表。每个报告都有一个结点值列表。

**示例 1：**



输入：[3,9,20,null,null,15,7]

输出：[[9],[3,15],[20],[7]]

解释：

在不丧失其普遍性的情况下，我们可以假设根结点位于 (0, 0)：

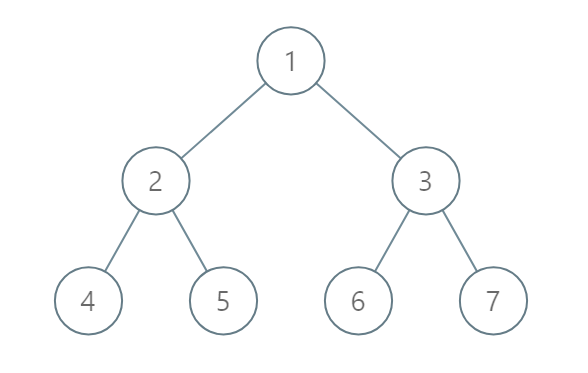
然后，值为 9 的结点出现在 (-1, -1)；

值为 3 和 15 的两个结点分别出现在 (0, 0) 和 (0, -2)；

值为 20 的结点出现在 (1, -1)；

值为 7 的结点出现在 (2, -2)。

**示例 2：**



输入：[1,2,3,4,5,6,7]

输出：[[4],[2],[1,5,6],[3],[7]]

解释：

根据给定的方案，值为 5 和 6 的两个结点出现在同一位置。

然而，在报告 "[1,5,6]" 中，结点值 5 排在前面，因为 5 小于 6。

提示：

树的结点数介于1和1000之间。

每个结点值介于0和1000之间。

# 分析

## 方法一：深度优先遍历

**思路：**

要按垂序遍历二叉树并返回结点值，可以使用广度优先搜索（BFS）和哈希表来实现。具体步骤如下：

1、创建一个哈希表 map<int, map<int, multiset<int>>>，其中外层 map 的键表示 X 坐标，内层 map 的键表示 Y 坐标，multiset 中存储了在同一位置的结点值，并且自动按照值的大小排序。

2、使用 BFS 遍历二叉树，并在遍历过程中记录每个结点的坐标。

3、将每个结点的值按照坐标存入哈希表中。

4、最后，按照 X 坐标从小到大遍历哈希表，将每个位置的结点值按照 Y 坐标和结点值的大小取出，加入结果集中。

**代码：**

class Solution {

public:

vector<vector<int>> verticalTraversal(TreeNode\* root) {

if (!root) return {};

map<int, map<int, multiset<int>>> nodes; // 哈希表记录每个结点的坐标和值

queue<pair<TreeNode\*, pair<int, int>>> q; // 队列用于 BFS

q.push({root, {0, 0}}); // 根节点坐标为 (0, 0)

while (!q.empty()) {

auto [node, coord] = q.front();

q.pop();

int x = coord.first, y = coord.second;

nodes[x][y].insert(node->val); // 插入结点值到对应坐标位置

if (node->left) q.push({node->left, {x - 1, y + 1}}); // 左子节点的坐标为 (x-1, y+1)

if (node->right) q.push({node->right, {x + 1, y + 1}}); // 右子节点的坐标为 (x+1, y+1)

}

vector<vector<int>> result;

for (auto& col : nodes) {

vector<int> col\_nodes;

for (auto& row : col.second) {

col\_nodes.insert(col\_nodes.end(), row.second.begin(), row.second.end());

}

result.push\_back(col\_nodes);

}

return result;

}

};

或：

class Solution {

public:

vector<vector<int>> verticalTraversal(TreeNode\* root) {

if (!root) return {};

map<int, map<int, multiset<int>>> nodes; // 哈希表记录每个结点的坐标和值

dfs(root, 0, 0, nodes); // 递归遍历二叉树

vector<vector<int>> result;

for (auto& col : nodes) {

vector<int> col\_nodes;

for (auto& row : col.second) {

col\_nodes.insert(col\_nodes.end(), row.second.begin(), row.second.end());

}

result.push\_back(col\_nodes);

}

return result;

}

void dfs(TreeNode\* node, int x, int y, map<int, map<int, multiset<int>>>& nodes) {

if (!node) return;

nodes[x][y].insert(node->val); // 插入结点值到对应坐标位置

dfs(node->left, x - 1, y + 1, nodes); // 递归遍历左子树，更新坐标

dfs(node->right, x + 1, y + 1, nodes); // 递归遍历右子树，更新坐标

}

};

**或：**

class Solution {

public:

vector<vector<int>> verticalTraversal(TreeNode\* root) {

// 存储节点的坐标和值

map<int, map<int, multiset<int>>> nodes;

// DFS遍历树，将节点的坐标和值存储到哈希表中

dfs(root, 0, 0, nodes);

// 将哈希表中的值按照题目要求组织成结果

vector<vector<int>> result;

for (auto& col : nodes) {

vector<int> colNodes;

for (auto& row : col.second) {

colNodes.insert(colNodes.end(), row.second.begin(), row.second.end());

}

result.push\_back(colNodes);

}

return result;

}

private:

void dfs(TreeNode\* node, int x, int y, map<int, map<int, multiset<int>>>& nodes) {

if (!node) return;

nodes[x][y].insert(node->val);

dfs(node->left, x - 1, y + 1, nodes);

dfs(node->right, x + 1, y + 1, nodes);

}

};