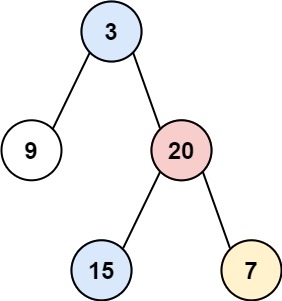
# 题目

给你一个二叉树的根结点，返回其结点按垂直方向（从上到下，逐列）遍历的结果。

如果两个结点在同一行和列，那么顺序则为从左到右。

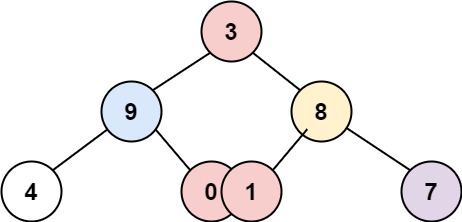
示例 1：



输入：root = [3,9,20,null,null,15,7]

输出：[[9],[3,15],[20],[7]]

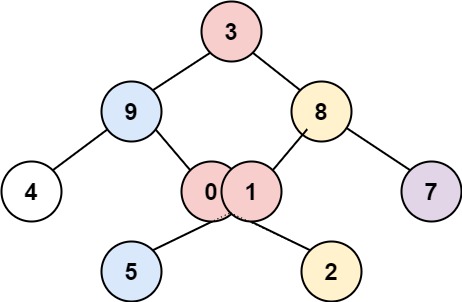
示例 2：



输入：root = [3,9,8,4,0,1,7]

输出：[[4],[9],[3,0,1],[8],[7]]

示例 3：



输入：root = [3,9,8,4,0,1,7,null,null,null,2,5]

输出：[[4],[9,5],[3,0,1],[8,2],[7]]

示例 4：

输入：root = []

输出：[]

提示：

树中结点的数目在范围 [0, 100] 内

-100 <= Node.val <= 100

类似题目987. Vertical Order Traversal of a Binary Tree

# 分析

## 方法一：深度优先遍历/DFS

**思路：**

总体思路是创建一个200 \* 100的二维矩阵，每个矩阵点要能够存放至少两个元素（可以用pair，或者再加一维矩阵构成三维矩阵）；

再先序遍历根节点，

mp[x][y].emplace\_back(root->val);

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

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

遍历完成之后，从三维矩阵中取出元素即可

**代码：**

class Solution {

public:

vector<vector<vector<int>>> mp;

void dfs(TreeNode \*root, int x, int y) {

if (root == nullptr) return;

mp[x][y].emplace\_back(root->val);

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

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

}

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

vector<vector<int>> ans;

if (root == nullptr) return ans;

mp.resize(202, vector<vector<int>>(101));

dfs(root, 101, 0);

for (int i = 0; i < 202; ++i) {

vector<int> cur;

for (int j = 0; j < 101; ++j) {

for (auto k : mp[i][j]) cur.emplace\_back(k);

}

if (cur.size() > 0) ans.emplace\_back(cur);

}

return ans;

}

};

BFS更加符合本题目。DFS略显臃肿，因为要记录层序号再排序才行。具体而言，DFS可以用先序遍历或者中序遍历树节点，保存节点值的深度和其值。排序是注意不要默认pair排序而是只按照第一维的深度排序就行了。相同深度可能有多个节点，不能排序，保持原来从左往右的顺序即可。

BFS思路类似，只是默认从上往下从左往右顺序遍历刚好符合要求，无需任何排序。天然适合本题。

代码：

class Solution {

public:

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

if (!root) {

return {};

}

dfs(root, 0, 0);

vector<vector<int>> res;

auto cmp = [](auto& a, auto& b) {

// Note: cannot use default comparison which is equivalent to the following,

// which also compares the second number, our case, when level is the same

// we keep the order from left to right, which is default in pre-order

// return a.first < b.first || (a.first == b.first && a.second < b.second);

return a.first < b.first;

};

for (auto& m : mp) {

sort(m.second.begin(), m.second.end(), cmp);

vector<int> col;

for (const auto& p : m.second) {

col.push\_back(p.second);

}

res.push\_back(col);

}

return res;

}

private:

map<int, vector<pair<int, int>>> mp;

// i is col index, while j is row index

void dfs(TreeNode\* root, int i, int j) {

if (!root) {

return;

}

mp[i].push\_back({j, root->val});

dfs(root->left, i - 1, j + 1);

dfs(root->right, i + 1, j + 1);

}

};

## 方法二：广度优先遍历/BFS

要解决二叉树的垂直遍历问题，核心是通过列索引标识每个节点的垂直位置，再按列收集节点值，同时保证“从上到下、同一行列内从左到右”的顺序。可以利用广度优先搜索（BFS）实现层序遍历，结合哈希表记录各列的节点值，最终按列索引排序输出结果。

思路：

1、列索引定义：

- 根节点的列索引为 0。

- 左子节点的列索引 = 父节点列索引 - 1（向左偏移一列）。

- 右子节点的列索引 = 父节点列索引 + 1（向右偏移一列）。

2、BFS 层序遍历：

- 使用队列存储“节点-列索引”对，确保遍历顺序是“从上到下”（层序）。

- 同一层内，左节点先入队、右节点后入队，确保“同一行列内从左到右”的顺序。

3、哈希表记录列数据：

- 用哈希表（键：列索引，值：该列的节点值列表）收集各列的节点值。

- 遍历结束后，按列索引从小到大排序，提取对应的值列表，即为垂直遍历结果。

代码：

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode() : val(0), left(nullptr), right(nullptr) {}

\* TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}

\* TreeNode(int x, TreeNode \*left, TreeNode \*right) : val(x), left(left), right(right) {}

\* };

\*/

class Solution {

public:

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

vector<vector<int>> result;

if (root == nullptr) return result;

// 哈希表：key=列索引，value=该列的节点值列表

unordered\_map<int, vector<int>> colMap;

// 队列：存储 (节点, 列索引) 对，用于BFS层序遍历

queue<pair<TreeNode\*, int>> q;

q.push({root, 0});

// 记录最小和最大列索引，避免后续遍历哈希表所有key（优化效率）

int minCol = 0, maxCol = 0;

while (!q.empty()) {

auto [currNode, currCol] = q.front(); // 取出队首节点及其列索引

q.pop();

// 将当前节点值加入对应列的列表

colMap[currCol].push\_back(currNode->val);

// 更新最小/最大列索引

minCol = min(minCol, currCol);

maxCol = max(maxCol, currCol);

// 左子节点入队（列索引-1），先左后右保证同一层顺序

if (currNode->left != nullptr) {

q.push({currNode->left, currCol - 1});

}

// 右子节点入队（列索引+1）

if (currNode->right != nullptr) {

q.push({currNode->right, currCol + 1});

}

}

// 按列索引从小到大，收集结果

for (int col = minCol; col <= maxCol; ++col) {

result.push\_back(colMap[col]);

}

return result;

}

};

代码解释：

1、初始化与边界处理：

- 若根节点为空，直接返回空结果。

- 队列初始化时存入根节点和其列索引 0；哈希表 colMap 用于按列存储节点值。

2、BFS 遍历过程：

- 每次取出队首节点，将其值加入对应列的列表，同时更新最小/最大列索引（避免后续遍历所有哈希表键）。

- 左子节点先入队（列索引-1），右子节点后入队（列索引+1），确保“同一层左到右”的顺序。

3、结果组装：

- 从最小列索引到最大列索引遍历，依次提取 colMap 中对应列的节点值列表，加入结果数组，保证“从左到右”的垂直顺序。

复杂度分析

- 时间复杂度：O(n log n)，其中 n 是二叉树节点数。

BFS 遍历所有节点：O(n)。

按列索引组装结果时，遍历范围是 [minCol, maxCol]，最坏情况下范围为 O(n)（如链状树），但无需排序哈希表键，实际效率接近 O(n)；若用普通哈希表且不记录 min/maxCol，需对键排序，复杂度为 O(k log k)（k为列数，k ≤ n），整体仍为 O(n log n)。

- 空间复杂度：O(n)。

队列最多存储一层节点，最坏情况下（满二叉树最后一层）为 O(n/2) = O(n)。

哈希表存储所有节点值，空间为 O(n)。

该方法通过 BFS 保证遍历顺序，结合哈希表高效分组，是解决二叉树垂直遍历的经典且高效的方案。