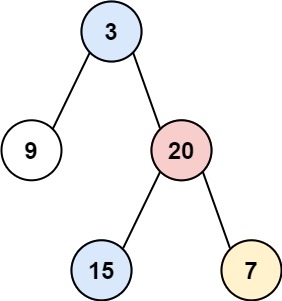
# 题目

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。

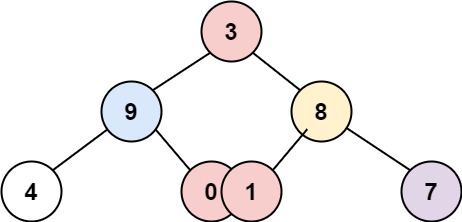
示例 1：



输入：root = [3,9,20,null,null,15,7]

输出：[[9],[3,15],[20],[7]]

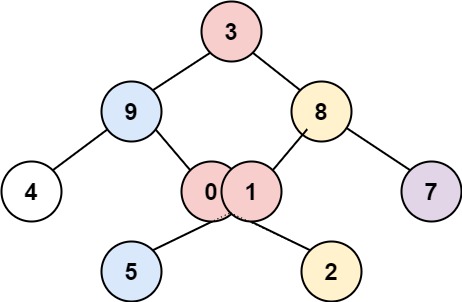
示例 2：



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

# 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

**思路：**

1、要求从上至下，那就BFS

2、要求从从至右

用结点的X坐标来分类

root的X坐标是0，所在竖线为y轴

**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)

{

if (root == NULL)

return {};

map<int, vector<int>> offset\_nodeVal; //带排序的字典

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

Q.emplace(root, 0);

while (Q.size())

{

auto [cur, offset] = Q.front();

Q.pop();

offset\_nodeVal[offset].push\_back(cur->val);

if (cur->left)

Q.emplace(cur->left, offset - 1); //offset以root所在的竖线为y轴，其他结点的x坐标

if (cur->right)

Q.emplace(cur->right, offset + 1);

}

vector<vector<int>> res;

for (auto [off, v]: offset\_nodeVal)

res.push\_back(v);

return res;

}

};

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

class Solution\_bfs {

public:

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

if (!root) {

return {};

}

map<int, vector<int>> mp;

queue<pair<int, TreeNode\*>> q{{make\_pair(0, root)}};

while (!q.empty()) {

for (int k = q.size(); k > 0; k--) {

auto cur = q.front(); q.pop();

mp[cur.first].push\_back(cur.second->val);

if (cur.second->left) {

q.push(make\_pair(cur.first - 1, cur.second->left));

}

if (cur.second->right) {

q.push(make\_pair(cur.first + 1, cur.second->right));

}

}

}

vector<vector<int>> res;

for (const auto& m : mp) {

res.push\_back(m.second);

}

return res;

}

};