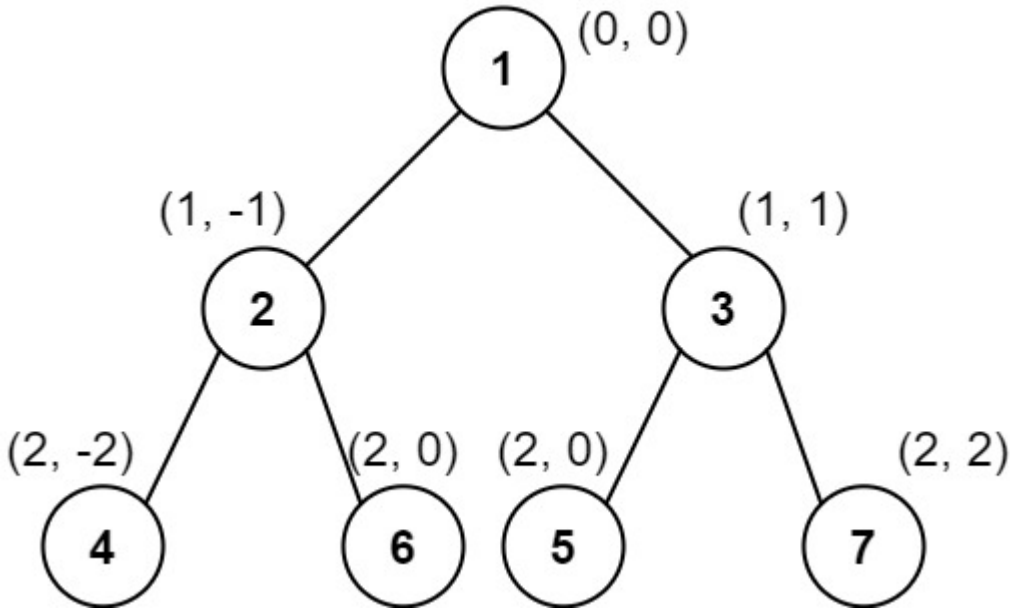
987.二叉树的垂序遍历

题目大意



记二叉树根节点坐标为 $(0, 0)$，对于每一个节点 (row, col)，其左儿子坐标为 $(row + 1, col - 1)$，其右儿子 $(row + 1, col + 1)$。从左到右，从上到下，排入节点的值，每一列打包为一个 `vector<int>` 对于坐标重叠的节点，按照节点的值从小到大排序。如图，输出应当为 `[4] [2] [1 5 6] [3] [7]`。

题目分析

我们考虑以每一列为单位，保存位于这一列里面的节点的值。观察到在最终的序列中，同一列内，深度潜的节点总是位于更前面，因此我们可以考虑层序 BFS ，每搜一层就把这一层的节点值 push 到对应的列中。
然而需要额外考虑到坐标重叠的情况：我们先把这一层的节点 push 到一个 buffer 里面，对每一列的 buffer 进行排序之后，再各自 push 到每一列中。

实现细节

由于 col 存在负值，因此不方便直接作为数组下标存储。好在树上节点个数 $n \leq 1000$，我们可以假装根节点的坐标为 $(0, 1001)$，把 col 数组大小开到 2005，时间复杂度为 $O(n)$， 具体代码如下：

```
struct Info {
    TreeNode* node;
    int nowcol;
    int nowrow;
};
```

```
vector< vector<int> > ret;
vector<int> column[2005];//col 0 = 1001, col -1000 = 1, col 1000 = 2001
vector<int> buffer[2005];
unordered_set<int> modified_id;
set<int> nonempty_col;
queue<Info> q;
void process_and_push() {
    for (auto onecol : modified_id) {
        sort(buffer[onecol].begin(), buffer[onecol].end());
        column[onecol].insert(column[onecol].end(),
                 buffer[onecol].begin(), buffer[onecol].end());
        buffer[onecol].clear();
    }
    modified_id.clear();
}
vector<vector<int>> verticalTraversal(TreeNode* root) {
    int bfsrow = 0;
    q.push({ root, 1001 });
    while (!q.empty()) {
        auto now = q.front();
        q.pop();
        if (bfsrow != now.nowrow) {
            process_and_push();
            bfsrow = now.nowrow;
        }
        nonempty_col.insert(now.nowcol);
        modified_id.insert(now.nowcol);
        buffer[now.nowcol].push_back(now.node->val);
        if (now.node->left)
                q.push({ now.node->left, (now.nowcol) - 1, (now.nowrow) + 1 });
        if (now.node->right)
                q.push({ now.node->right, (now.nowcol) + 1, (now.nowrow) + 1 });
    }
    process_and_push();
    for (auto nowcol : nonempty_col) {
        ret.push_back(column[nowcol]);
    }
    return ret;
}
```