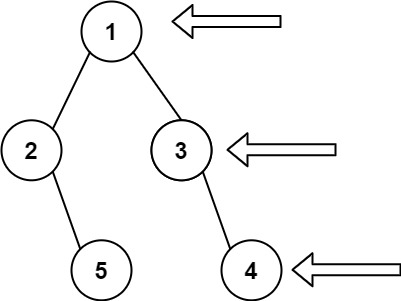
# 题目

给定一个二叉树的 根节点root，想象自己站在它的右侧，按照从顶部到底部的顺序，返回从右侧所能看到的节点值。

示例 1:



输入: [1,2,3,null,5,null,4]

输出: [1,3,4]

示例 2:

输入: [1,null,3]

输出: [1,3]

示例 3:

输入: []

输出: []

提示:

二叉树的节点个数的范围是 [0,100]

-100 <= Node.val <= 100

注意：本题与主站199 题相同：

https://leetcode-cn.com/problems/binary-tree-right-side-view/

# 分析

/\*\*

 \* 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<int> rightSideView(TreeNode\* root) {

        unordered\_map<int, int> rightmostValueAtDepth;

        int max\_depth = -1;

        queue<TreeNode\*> nodeQueue;

        queue<int> depthQueue;

        nodeQueue.push(root);

        depthQueue.push(0);

        while (!nodeQueue.empty()) {

            TreeNode\* node = nodeQueue.front();nodeQueue.pop();

            int depth = depthQueue.front();depthQueue.pop();

            if (node != NULL) {

                // 维护二叉树的最大深度

                max\_depth = max(max\_depth, depth);

                // 由于每一层最后一个访问到的节点才是我们要的答案，因此不断更新对应深度的信息即可

                rightmostValueAtDepth[depth] =  node -> val;

                nodeQueue.push(node -> left);

                nodeQueue.push(node -> right);

                depthQueue.push(depth + 1);

                depthQueue.push(depth + 1);

            }

        }

        vector<int> rightView;

        for (int depth = 0; depth <= max\_depth; ++depth) {

            rightView.push\_back(rightmostValueAtDepth[depth]);

        }

        return rightView;

    }

};