# 二叉树的右视图

给定一棵二叉树，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:**

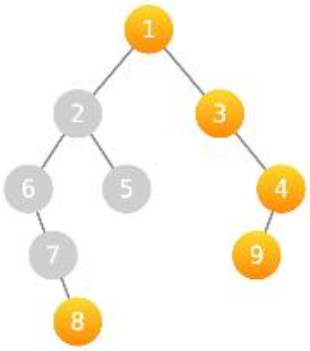
**输入:** [1,2,3,null,5,null,4]

**输出:** [1, 3, 4]

**解释:**

```
   1            <---
 /   \
2     3         <---
 \     \
  5     4       <---
```

- 方法 2：深度优先搜索
  - 我们按照 **根结点 -> 右子树 -> 左子树** 的顺序访问，可以保证每层都是最先访问最右边的节点。




result = [1, 3, 4, 9, 8]

```
/**
 * 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:
    //全局变量
    vector<int> res;
    void dfs(TreeNode* root,int depth)
    {
        if(root==nullptr)
            return;
        if(depth==res.size())//访问当前节点 判断当前节点所在层深度与容器大小的关系，相等即为该深度下第一个遍历到的节点
        {
            //如果当前节点所在深度还没有放入结果容器中 说明为每一层第一个遍历到的节点，即为结果
            res.push_back(root->val);
        }
        depth++;
        dfs(root->right,depth);//递归调用右子树
        dfs(root->left,depth);//递归调用左子树
    }
    vector<int> rightSideView(TreeNode* root) {
        //根  右  左
        //保证每层第一个访问到的即为结果
        dfs(root,0);//从根节点开始访问，深度初始为0  结果容器大小也为0
        return res;
    }
};
```