从上到下打印二叉树

> `dfs` 二叉树的遍历

```
class Solution {
public:
    // 题目要求的二叉树的 从上至下 打印（即按层打印），
    // 又称为二叉树的 广度优先搜索（BFS）
    vector<int> levelOrder(TreeNode *root) {
        vector<int> ans{};
        if(root== nullptr){
            return ans;
        }
        queue<TreeNode* > queueTree;
        queueTree.push(root);
        while (!queueTree.empty()){
            TreeNode* node = queueTree.front();
            queueTree.pop();
            ans.push_back(node->val);
            if(node->left!= nullptr){
                queueTree.push(node->left);
            }
            if(node->right!= nullptr){
                queueTree.push(node->right);
            }
        }
        return ans;
    }
};
```

从上到下打印二叉树 II

```
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        vector<vector<int>> ans;
        if(root== nullptr){
            return  vector<vector<int>>{};
        }
        queue<TreeNode* > queueTree;
        queueTree.push(root);
        while (!queueTree.empty()){
            vector<int> tmp;
            for (int i = queueTree.size(); i >0 ; --i) {
                TreeNode* node = queueTree.front();
                queueTree.pop();
                tmp.push_back(node->val);
                if(node->left!= nullptr){
                    queueTree.push(node->left);
                }
                if(node->right!= nullptr){
                    queueTree.push(node->right);
```

```
                }
            }
            ans.push_back(tmp);
        }
        return ans;
    }
};
```

从上到下打印二叉树 III

```
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode *root) {
        vector<vector<int>> ans;
        if (root == nullptr) {
            return vector<vector<int>>{};
        }
        queue<TreeNode *> queueTree;
        queueTree.push(root);
        while (!queueTree.empty()) {
            vector<int> tmp;
            for (int i = queueTree.size(); i > 0; --i) {
                TreeNode *node = queueTree.front();
                queueTree.pop();
                if(ans.size() % 2 == 0)
                {
                    tmp.push_back(node->val); // 偶数层 -> 队列头部
                }else{
                    tmp.insert(tmp.begin(),node->val);
                }
                if (node->left != nullptr) {
                    queueTree.push(node->left);
                }
                if (node->right != nullptr) {
                    queueTree.push(node->right);
                }
            }
            ans.push_back(tmp);
        }
        return ans;
    }
};
```