144. 二叉树的前序遍历

https://leetcode-cn.com/problems/binary-tree-preorder-traversal/

```
class Solution {
public:
    // 求出以root为根的树的前序遍历
    void preorder(TreeNode *root, vector<int>& order) {
        if (root == nullptr) return ;
        // 根
        order.push_back(root->val);
        // 左子树
        preorder(root->left, order);
        // 右子树
        preorder(root->right, order);
    }

    vector<int> preorderTraversal(TreeNode* root) {
        vector<int> order;
        preorder(root, order);
        return order;
    }
};
```

589. N 叉树的前序遍历

https://leetcode-cn.com/problems/n-ary-tree-preorder-traversal/

```
class Solution {
public:
    void _preorder(Node *root, vector<int>& order) {
        if (root == nullptr) return;
        order.push_back(root->val);
        for (int i = 0; i < root->children.size(); i++) {
            _preorder(root->children[i], order);
        }
    }

    vector<int> preorder(Node* root) {
        vector<int> order;
        _preorder(root, order);
        return order;
    }
};
```

226. 翻转二叉树

https://leetcode-cn.com/problems/invert-binary-tree/

```
class Solution {
public:
    TreeNode* invertTree(TreeNode* root) {
        if (!root) return nullptr;
        swap(root->left, root->right);
        invertTree(root->left);
        invertTree(root->right);
        return root;
    }
};
```

102. 二叉树的层序遍历

https://leetcode-cn.com/problems/binary-tree-level-order-traversal/

```
class Solution {
public:
    void _levelOrder(TreeNode *root, int depth, vector<vector<int>> &ans) {
        if (!root) return ;
        if (depth - 1 == ans.size()) ans.push_back(vector<int> ());
        ans[depth - 1].push_back(root->val);
        _levelOrder(root->left, depth + 1, ans);
        _levelOrder(root->right, depth + 1, ans);
    }

    vector<vector<int>> levelOrder(TreeNode* root) {
        vector<vector<int>> ans;
        _levelOrder(root, 1, ans);
        return ans;
    }
};
```

```
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        vector<vector<int>> ans;
        if (root == nullptr) return ans;
        queue<TreeNode*> q;
        q.push(root);
        while (!q.empty()) {
```

```
            int qSize = q.size();
            ans.push_back(vector<int> ());
            for (int i = 0; i < qSize; i++) {
                TreeNode *node = q.front();
                q.pop();
                ans.back().push_back(node->val);
                if (node->left) q.push(node->left);
                if (node->right) q.push(node->right);
            }
        }
        return ans;
    }
};
```

107. 二叉树的层序遍历 II

https://leetcode-cn.com/problems/binary-tree-level-order-traversal-ii/.

```
class Solution {
public:
    void _levelOrder(TreeNode *root, int depth, vector<vector<int>> &ans) {
        if (!root) return ;
        if (depth - 1 == ans.size()) ans.push_back(vector<int> ());
        ans[depth - 1].push_back(root->val);
        _levelOrder(root->left, depth + 1, ans);
        _levelOrder(root->right, depth + 1, ans);
    }

    vector<vector<int>> levelOrderBottom(TreeNode* root) {
        vector<vector<int>> ans;
        _levelOrder(root, 1, ans);
        for (int i = 0; i < ans.size() / 2; i++) {
            swap(ans[i], ans[ans.size() - i - 1]);
        }
        return ans;
    }
};
```

103. 二叉树的锯齿形层序遍历

https://leetcode-cn.com/problems/binary-tree-zigzag-level-order-traversal/

```
class Solution {
public:
```

```
    void _levelOrder(TreeNode *root, int depth, vector<vector<int>> &ans) {
        if (!root) return ;
        if (depth - 1 == ans.size()) ans.push_back(vector<int> ());
        ans[depth - 1].push_back(root->val);
        _levelOrder(root->left, depth + 1, ans);
        _levelOrder(root->right, depth + 1, ans);
    }

    vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
        vector<vector<int>> ans;
        _levelOrder(root, 1, ans);
        for (int i = 1; i < ans.size(); i += 2) {
            for (int j = 0; j < ans[i].size() / 2; j++) {
                swap(ans[i][j], ans[i][ans[i].size() - j - 1]);
            }
        }
        return ans;
    }
};
```

110. 平衡二叉树

https://leetcode-cn.com/problems/balanced-binary-tree/

```
class Solution {
public:
    int getHeight(TreeNode *root) {
        if (root == nullptr) return 0;
        int l = getHeight(root->left);
        int r = getHeight(root->right);
        if (l == -1 || r == -1) return -1;
        if (abs(l - r) > 1) return -1;
        return max(l, r) + 1;
    }

    bool isBalanced(TreeNode* root) {
        return getHeight(root) != -1;
    }
};
```

112. 路径总和

https://leetcode-cn.com/problems/path-sum/

```cpp
class Solution {
public:
    bool hasPathSum(TreeNode* root, int targetSum) {
        if (!root) return false;
        if (!root->left && !root->right) return root->val == targetSum;
        bool l = hasPathSum(root->left, targetSum - root->val);
        bool r = hasPathSum(root->right, targetSum - root->val);
        return l || r;
    }
};
```

105. 从前序与中序遍历序列构造二叉树

https://leetcode-cn.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/

```cpp
class Solution {
public:
    TreeNode *build(vector<int> &preorder, int preL, int preR,
                    vector<int> &inorder,  int inL,  int inR) {
        if (preL > preR || inL > inR) return nullptr;
        TreeNode *root = new TreeNode(preorder[preL]);
        int idx = inL;
        for (; idx <= inR; idx++) {
            if (inorder[idx] == root->val) break;
        }
        root->left = build(preorder, preL + 1, preL + (idx - inL), inorder, inL, idx - 1);
        root->right = build(preorder, preL + (idx - inL) + 1, preR, inorder, idx + 1, inR);
        return root;
    }

    TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {
        return build(preorder, 0, preorder.size() - 1, inorder, 0, inorder.size() - 1);
    }
};
```

222. 完全二叉树的节点个数

https://leetcode-cn.com/problems/count-complete-tree-nodes/

```
class Solution {
public:
    int getHeight(TreeNode *root) {
        TreeNode *p = root;
        int height = 0;
        while (p) {
            p = p->left;
            height++;
        }
        return height;
    }

    int countNodes(TreeNode* root) {
        if (!root) return 0;
        int l = getHeight(root->left);
        int r = getHeight(root->right);
        if (l > r) return countNodes(root->left) + (1 << r);
        else return countNodes(root->right) + (1 << l);
    }
};
```