# 二叉树专题（labuladong）

## 144、94、145二叉树的前中后序遍历

递归写法好说

注意掌握迭代写法

基本思路是写一个栈，把根节点入栈，然后循环遍历取栈顶，注意几点：

1. 子节点入栈的顺序是先右后左
2. 根节点进入ans vector
3. 前中后顺序不同
4. 中序和后序遍历要加flag，只有回溯的时候才把根节点加入ans vector

有一种方式是压空包，我实现的另一种方式是搞一个std::pair，都行

## 102 二叉树的层序遍历

While(q)里面做一些细的操作，先取得目前队列的长度，然后for循环，这样可以一层一层地分开遍历

## 104 二叉树最大深度

自顶向下和自底向上，两种思路

// LMB : 自顶向下

class Solution {

public:

int maxDepth(TreeNode\* root) {

int max\_depth = 0;

int depth = 0;

depthHelper(root, depth, max\_depth);

return max\_depth;

}

void depthHelper(TreeNode\* root, int depth, int& max\_depth) {

if(!root) return;

if(!root->left && !root->right) {

max\_depth = max(max\_depth, depth+1);

}

depthHelper(root->left, depth+1, max\_depth);

depthHelper(root->right, depth+1, max\_depth);

}

};

 // LMB : 自底向上

class Solution {

public:

    int maxDepth(TreeNode\* root) {

        if(!root) return 0;

        return 1 + max(maxDepth(root->left), maxDepth(root->right));

    }

};

或者

class Solution {

public:

int maxDepth(TreeNode\* root) {

return root ? 1 + max(maxDepth(root->left), maxDepth(root->right)) : 0;

}

};

## 101 对称二叉树

注意不能从根节点写一个函数来决定，比如下面这样是错的！

class Solution {

public:

    bool isSymmetric(TreeNode\* root) {

        if(!root) return true;

        if(!root->left && !root->right) return true;

        if(!root->left || !root->right) return false;

        if(root->left->val != root->right->val) return false;

        return isSymmetric(root->left) && isSymmetric(root->right);

    }

};

必须写一个双节点递归函数才行：

class Solution {

public:

bool isSymmetric(TreeNode\* root) {

return root ? isSymmetric(root->left, root->right) : true;

}

bool isSymmetric(TreeNode\* left, TreeNode\* right) {

if( !left && !right ) return true;

if( !left || !right ) return false;

if(left->val != right->val) return false;

return isSymmetric(left->left, right->right) && isSymmetric(left->right, right->left);

}

};

## 112、路径总和

class Solution {

public:

    bool hasPathSum(TreeNode\* root, int sum) {

        if(!root) return false;

        if(!root->left && !root->right) return root->val == sum;

        return hasPathSum(root->left, sum-root->val) || hasPathSum(root->right, sum-root->val);

    }

}

注意分析根节点为空 与 叶节点的情形能不能合并。本题不能合并，所以要分开写！

## 116. 填充每个节点的下一个右侧节点指针

以前，这道题我用层序遍历来生硬解决：

class Solution {

public:

Node\* connect(Node\* root) {

if(!root) return nullptr;

queue<Node\*> q;

q.push(root);

while(!q.empty()) {

int count = q.size();

Node\* leftest = q.front(), \*prev = leftest;

q.pop();

if(leftest->left) q.push(leftest->left);

if(leftest->right) q.push(leftest->right);

for(int i=1; i<count; ++i) {

Node\* node = q.front();

q.pop();

prev->next = node;

prev = node;

if(node->left) q.push(node->left);

if(node->right) q.push(node->right);

}

prev->next = nullptr;

}

return root;

}

};

现在，我用递归来解决。注意，要启用双节点递归：

class Solution {

public:

    Node\* connect(Node\* root) {

        if(root) connect(root->left, root->right);

        return root;

    }

    void connect(Node\* left, Node\* right) {

        if(!left || !right) return;

        left->next = right;

        connect(left->left, left->right);

        connect(right->left, right->right);

        connect(left->right, right->left);

    }

};

## 226、翻转二叉树

class Solution {

public:

    TreeNode\* invertTree(TreeNode\* root) {

        helper(root);

        return root;

    }

    void helper(TreeNode\* root) {

        if(!root) return;

        swap(root->left, root->right);

        helper(root->left);

        helper(root->right);

    }

};

## 114、二叉树展开为链表

class Solution {

public:

    void flatten(TreeNode\* root) {

        if(!root) return;

        // if(!root->left && !root->right) return;

        flatten(root->left);

        flatten(root->right);

        TreeNode\* left\_raw = root->left, \*right\_raw = root->right;

        root->left = nullptr;

        root->right = left\_raw;

        TreeNode\* n=root;

        while(n->right != nullptr) {

            n = n->right;

        }

        n->right = right\_raw;

    }

};

思路：典型的后续遍历

## 206 反转链表

迭代写法：

class Solution {

public:

ListNode\* reverseList(ListNode\* head) {

ListNode \*prev = nullptr, \*next;

while(head) {

next = head->next;

head->next = prev;

prev = head;

head = next;

}

return prev;

}

};

后序遍历递归写法（回溯写法）

// 回溯写法

class Solution {

public:

ListNode\* reverseList(ListNode\* head) {

if( !head || !head->next ) {

return head;

}

ListNode\* ret = reverseList(head->next);

guard->next->next = head;

head->next = nullptr;

return ret;

}

};

先序遍历，双参数写法：

class Solution {

public:

// ListNode\* reverseList(ListNode\* head) {

// return reverseListRecursive(head);

// }

ListNode\* reverseList(ListNode\* head, ListNode\* prev = nullptr) {

if(!head) {

return prev;

}

ListNode\* next = head->next;

head->next = prev;

return reverseList(next, head);

}

};