# 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

# 翻转链表（反转链表）专题

## 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);

head->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);

}

};

## 92 反转链表 II

// 方法源自labuladong

/\*

https://mp.weixin.qq.com/s?\_\_biz=MzAxODQxMDM0Mw==&mid=2247484467&idx=1&sn=beb3ae89993b812eeaa6bbdeda63c494&chksm=9bd7fa3baca0732dc3f9ae9202ecaf5c925b4048514eeca6ac81bc340930a82fc62bb67681fa&scene=21#wechat\_redirect

\*/

class Solution {

public:

    ListNode\* reverseBetween(ListNode\* head, int m, int n) {

        if( m == 1 ) {

            ListNode\* successor = nullptr;

            return reverseN(head, n, successor);

        }

        head->next = reverseBetween(head->next, m-1, n-1);

        return head;

    }

    ListNode\* reverseN(ListNode\* head, int n, ListNode\*& successor) {

        if(n==0) return head;

        if(n==1) {

            successor = head->next;

            return head;

        }

        ListNode\* ret = reverseN(head->next, n-1, successor);

        head->next->next = head;

        head->next = successor;

        return ret;

    }

};

关键：学会用一句或两句白话来说清楚函数的作用，并且坚决相信函数可以实现它。

值得一提的是，**递归操作链表并不高效。**和迭代解法相比，虽然时间复杂度都是 O(N)，但是迭代解法的空间复杂度是 O(1)，而递归解法需要堆栈，空间复杂度是 O(N)。所以递归操作链表可以作为对递归算法的练习或者拿去和小伙伴装逼，但是考虑效率的话还是使用迭代算法更好。

## 25 K个一组翻转链表

class Solution {

public:

    ListNode\* reverseN(ListNode\* a, ListNode\* b) {

        ListNode \*prev = nullptr, \*next;

        while(a!=b) {

            next = a->next;

            a->next = prev;

            prev = a;

            a = next;

        }

        return prev;

    }

    ListNode\* reverseKGroup(ListNode\* head, int k) {

        ListNode \*a = head, \*b = head;

        for(int i=0; i<k; ++i) {

            if( !b ) return head;

            b = b->next;

        }

        ListNode\* ret = reverseN(a, b);

        a->next = reverseKGroup(b, k);

        return ret;

    }

};

/\*

参考labuladong

https://mp.weixin.qq.com/s?\_\_biz=MzAxODQxMDM0Mw==&mid=2247484597&idx=1&sn=c603f1752e33cb2701e371d84254aee2&chksm=9bd7fabdaca073abd512d8fff18016c9092ede45fed65c307852c65a2026d8568ee294563c78&scene=21#wechat\_redirect

思路 ：外层函数定义，K个一组翻转以head为头结点的链表，返回翻转后的头结点

一般情形：

1、翻转前k个节点

2、K个一组翻转以head往右k个结点为头结点的链表，返回翻转后的头结点（子问题）

3、两者接起来

base case：

不足k个结点，直接返回head

\*/

心得：没想清楚思路的时候，要做的就是：

甩开编程语言，用白话去分析问题

怎么用白话分析问题？清晰地用语言描述函数的功能。

怎么清晰地用语言描述函数的功能？输入什么，做什么事，返回什么

其实不只是编程，生活中的任何事都是两点：

1. 想清楚；2、干明白

想清楚是干明白的前提

事情进展不顺的时候，要像打篮球那样，叫个暂停

分析下究竟是没想清楚，还是没干明白

# 回文小专题

回文是什么？是正着读反着读都一样的字符串。解决回文问题的核心是双指针

英文名：**Palindrome**

## 25 最长回文子串

思路：

**for** 0 <= i < len(s):  
    找到以 s[i] 为中心的回文串  
    找到以 s[i] 和 s[i+1] 为中心的回文串  
    更新答案

class Solution {

public:

    string longestPalindrome(string s) {

        string res;

        for(int i=0; i<s.length(); ++i) {

            string s1 = palindrome(s, i, i);

            string s2 = palindrome(s, i, i+1);

            res = s1.length() > res.length() ? s1 : res;

            res = s2.length() > res.length() ? s2 : res;

        }

        return res;

    }

    string palindrome(string s, int l, int r) {

        while(l>=0 && r<s.length() && s[l]==s[r]) {

            --l;

            ++r;

        }

        return s.substr(l+1, (r-1)-(l+1)+1);

    }

};

/\*

来自labuladong

https://mp.weixin.qq.com/s?\_\_biz=MzAxODQxMDM0Mw==&mid=2247484471&idx=1&sn=7c26d04a1f035770920d31377a1ebd42&chksm=9bd7fa3faca07329189e9e8b51e1a665166946b66b8e8978299ba96d5f2c0d3eafa7db08b681&scene=21#wechat\_redirect

思路：

int longestPalindrome(string s) {

    for i 0~len

        pnlindrome(s, i, i)

        palindrome(s, i, i+1)

        更新

}

\*/

## 234 回文链表

方法一：后序遍历

class Solution {public:

bool isPalindrome(ListNode\* head) {

ListNode\* left = head;

return traverse(left, head);

}

bool traverse(ListNode\*& left, ListNode\* right) {

if(!right) return true;

bool ret = traverse(left, right->next);

ret = ret && (left->val == right->val);

left = left->next;

return ret;

}

};

/\*

后续遍历

过程略有冗余，来自labuladong

https://mp.weixin.qq.com/s?\_\_biz=MzAxODQxMDM0Mw==&mid=2247484822&idx=1&sn=44742c9a3557038c8da7150100d94db9&chksm=9bd7fb9eaca0728876e1146306a09f5453bcd5c35c4a264304ea6189faa83ec12a00322f0246&scene=21#wechat\_redirect

\*/

方法二：快慢指针找中点 + 反转

class Solution {

public:

    bool isPalindrome(ListNode\* head) {

        ListNode \*fast = head, \*slow = head;

        while(fast && fast->next) {

            fast = fast->next->next;

            slow = slow->next;

        }

        if(fast) slow = slow->next;

        ListNode \*left = head, \*right = reverse(slow);

        while(right) {

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

            left = left->next;

            right = right->next;

        }

        return true;

    }

    ListNode\* reverse(ListNode\* head) {

        ListNode \*prev = nullptr, \*next;

        while(head) {

            next = head->next;

            head->next = prev;

            prev = head;

            head = next;

        }

        return prev;

    }

};

/\*

先用快慢指针找中点的方法

来自labuladong

https://mp.weixin.qq.com/s?\_\_biz=MzAxODQxMDM0Mw==&mid=2247484822&idx=1&sn=44742c9a3557038c8da7150100d94db9&chksm=9bd7fb9eaca0728876e1146306a09f5453bcd5c35c4a264304ea6189faa83ec12a00322f0246&scene=21#wechat\_redirect

\*/

# 回到二叉树

## 235 二叉树的最近公共祖先

这道题代码看着简单，其实非常不好理解。但它却是理解二叉树遍历解法的一道非常好的题目。

class Solution {

public:

    TreeNode\* lowestCommonAncestor(TreeNode\* root, TreeNode\* p, TreeNode\* q) {

        if(!root) return nullptr;

        if(p==root || q==root) return root;

        TreeNode\* left = lowestCommonAncestor(root->left, p, q);

        TreeNode\* right = lowestCommonAncestor(root->right, p, q);

        if(left != nullptr && right != nullptr) {

            return root;

        }

        if(left == nullptr && right == nullptr) {

            return nullptr;

        }

        return !left ? right : left;

    }

};

/\*

来自labuladong

https://mp.weixin.qq.com/s?\_\_biz=MzAxODQxMDM0Mw==&mid=2247485561&idx=1&sn=a394ba978283819da1eb34a256f6915b&chksm=9bd7f671aca07f6722f0bc1e946ca771a0a40fd8173cc1227a7e0eabfe4e2fcc57b9ba464547&scene=21#wechat\_redirect

这是一道很经典的二叉树递归的题目，非常非常经典。

\*/

## 654 最大二叉树

比较简单：

class Solution {

public:

    TreeNode\* constructMaximumBinaryTree(vector<int>& nums) {

        return build(nums, 0, nums.size()-1);

    }

    TreeNode\* build(vector<int>& nums, int l, int r) {

        if(l>r) return nullptr;

        int max\_idx = -1;

        int max\_val = INT\_MIN;

        for(int i=l; i<=r; ++i) {

            if(nums[i] > max\_val) {

                max\_idx = i;

                max\_val = nums[i];

            }

        }

        TreeNode\* root = new TreeNode(max\_val);

        root->left = build(nums, l, max\_idx-1);

        root->right = build(nums, max\_idx+1, r);

        return root;

    }

};

## 105 从前序与中序遍历序列构造二叉树

class Solution {

public:

    TreeNode\* buildTree(vector<int>& preorder, vector<int>& inorder) {

        return build(preorder, 0, preorder.size()-1, inorder, 0, inorder.size()-1);

    }

    TreeNode\* build(vector<int>& preorder, int pre\_l, int pre\_r,

                    vector<int>& inorder, int in\_l, int in\_r) {

        if(pre\_l > pre\_r) return nullptr;

        int root\_val = preorder[pre\_l];

        int root\_idx\_inorder = -1;

        for(int i=in\_l; i<=in\_r; ++i) {

            if(inorder[i] == root\_val) {

                root\_idx\_inorder = i;

            }

        }

        int left\_num = root\_idx\_inorder - in\_l;

        // int right\_num = in\_r - root\_idx\_inorder;

        TreeNode\* root = new TreeNode(root\_val);

        root->left = build(preorder, pre\_l+1, pre\_l+left\_num, inorder, in\_l, root\_idx\_inorder-1);

        root->right = build(preorder, pre\_l+left\_num+1, pre\_r, inorder, root\_idx\_inorder+1, in\_r);

        return root;

    }

};

// 要完全理解才行

## 106 从中序与后序遍历序列构造二叉树

class Solution {

public:

TreeNode\* buildTree(vector<int>& inorder, vector<int>& postorder) {

return build(inorder, 0, inorder.size()-1, postorder, 0, postorder.size()-1);

}

TreeNode\* build(vector<int>& inorder, int in\_l, int in\_r,

vector<int>& postorder, int post\_l, int post\_r) {

if(post\_l > post\_r) return nullptr;

int root\_val = postorder[post\_r];

int root\_index;

TreeNode\* root = new TreeNode(root\_val);

for(int i=in\_l; i<=in\_r; ++i) {

if(inorder[i] == root\_val) {

root\_index = i;

break;

}

}

int right\_num = in\_r - root\_index;

root->right = build(inorder, root\_index+1, in\_r, postorder, post\_r-1-right\_num+1, post\_r-1);

root->left = build(inorder, in\_l, root\_index-1, postorder, post\_l, post\_r-1-right\_num);

return root;

}

};

这个注意子树是先右后左即可。其他的就是计算index的细节。

## 297 二叉树的序列化与反序列化

前序遍历、后序遍历、层序遍历三种解法，看提交记录即可

// 前序遍历法

class Codec {

public:

// Encodes a tree to a single string.

string serialize(TreeNode\* root) {

string ret = "[";

ret += serializeHelper(root);

ret.back() = ']';

// cout << ret << endl;

return ret;

}

string serializeHelper(TreeNode\* root) {

if(!root) return "null,";

string ret = to\_string(root->val) + ",";

ret += serializeHelper(root->left);

ret += serializeHelper(root->right);

return ret;

}

// Decodes your encoded data to tree.

TreeNode\* deserialize(string data) {

queue<int> nodes;

decode(data, nodes);

return deserializeHelper(nodes);

}

TreeNode\* deserializeHelper(queue<int>& nodes) {

if(nodes.empty()) return nullptr;

int root\_val = nodes.front();

nodes.pop();

if(root\_val == INT\_MIN) return nullptr;

TreeNode\* root = new TreeNode(root\_val);

root->left = deserializeHelper(nodes);

root->right = deserializeHelper(nodes);

return root;

}

void decode(const string& code, queue<int>& nodes) {

string \_code = code.substr(1, code.length()-2);

// cout << \_code << endl;

istringstream iss(\_code);

string line;

while(getline(iss, line, ',')) {

int val = line == "null" ? INT\_MIN : stod(line);

nodes.push(val);

}

}

};

// 后序遍历法

class Codec {

public:

// Encodes a tree to a single string.

string serialize(TreeNode\* root) {

string ret = "[";

ret += serializeHelper(root);

ret.back() = ']';

return ret;

}

string serializeHelper(TreeNode\* root) {

if(!root) return "null,";

string ret;

ret += serializeHelper(root->left);

ret += serializeHelper(root->right);

ret += to\_string(root->val) + ",";

return ret;

}

// Decodes your encoded data to tree.

TreeNode\* deserialize(string data) {

stack<int> nodes;

decode(data, nodes);

return deserializeHelper(nodes);

}

TreeNode\* deserializeHelper(stack<int>& nodes) {

if(nodes.empty()) return nullptr;

int root\_val = nodes.top();

nodes.pop();

if(root\_val == INT\_MIN) return nullptr;

TreeNode\* root = new TreeNode(root\_val);

root->right = deserializeHelper(nodes);

root->left = deserializeHelper(nodes);

return root;

}

void decode(const string& code, stack<int>& nodes) {

string \_code = code.substr(1, code.length()-2);

istringstream iss(\_code);

string line;

while(getline(iss, line, ',')) {

int val = line == "null" ? INT\_MIN : stod(line);

nodes.push(val);

}

}

};

// 层序遍历法

// 方法三 ： 层序遍历

class Codec {

public:

// Encodes a tree to a single string.

string serialize(TreeNode\* root) {

string ret = "[";

ret += serializeHelper(root);

ret.back() = ']';

return ret;

}

string serializeHelper(TreeNode\* root) {

if(!root) return "null,";

string ret;

queue<TreeNode\*> q;

q.push(root);

while(!q.empty()) {

TreeNode\* node = q.front();

q.pop();

if(!node) {

ret += "null,";

continue;

}

ret += to\_string(node->val) + ",";

q.push(node->left);

q.push(node->right);

}

cout << ret << endl;

return ret;

}

// Decodes your encoded data to tree.

TreeNode\* deserialize(string data) {

queue<int> nodes;

decode(data, nodes);

return deserializeHelper(nodes);

}

TreeNode\* deserializeHelper(queue<int>& nodes) {

if(nodes.empty()) return nullptr;

if(nodes.front() == INT\_MIN) return nullptr;

TreeNode\* root = new TreeNode(nodes.front());

nodes.pop();

queue<TreeNode\*> q;

q.push(root);

while(!nodes.empty()) {

TreeNode\* farther = q.front();

q.pop();

if(!farther) continue;

int left\_val = nodes.front();

nodes.pop();

if(left\_val != INT\_MIN) {

TreeNode\* left\_child = new TreeNode(left\_val);

farther->left = left\_child;

q.push(left\_child);

} else {

farther->left = nullptr;

}

int right\_val = nodes.front();

nodes.pop();

if(right\_val != INT\_MIN) {

TreeNode\* right\_child = new TreeNode(right\_val);

farther->right = right\_child;

q.push(right\_child);

} else {

farther->right = nullptr;

}

}

return root;

}

void decode(const string& code, queue<int>& nodes) {

string \_code = code.substr(1, code.length()-2);

istringstream iss(\_code);

string line;

while(getline(iss, line, ',')) {

int val = line == "null" ? INT\_MIN : stod(line);

nodes.push(val);

}

}

};

## 652 寻找重复的子树

难度2

class Solution {

public:

    vector<TreeNode\*> findDuplicateSubtrees(TreeNode\* root) {

        vector<TreeNode\*> result;

        map<string, int> subtree\_counter;

        traverse(root, subtree\_counter, result);

        return result;

    }

    // 序列化函数

    // traverse函数

    string traverse(TreeNode\* root, map<string, int>& subtree\_counter, vector<TreeNode\*>& result) {

        if(!root) return "";

        string left = traverse(root->left, subtree\_counter, result);

        string right = traverse(root->right, subtree\_counter, result);

        string subtree = left + "," + right + "," + to\_string(root->val);

        ++subtree\_counter[subtree];

        if(subtree\_counter[subtree] == 2) {

            result.push\_back(root);

        }

        return subtree;

    }

};

## 230 二叉搜索树中第k小的元素

难度1

class Solution {

public:

    int kthSmallest(TreeNode\* root, int k) {

        int rank = 0, result = 0;

        traverse(root, k, rank, result);

        return result;

    }

    void traverse(TreeNode\* root, int k, int& rank, int& result) {

        if(!root) return;

        traverse(root->left, k, rank, result);

        ++rank;

        if(rank == k) {

            result = root->val;

            return;

        }

        traverse(root->right, k, rank, result);

    }

};

## 538 把二叉搜索树转换为累加树

难度2

class Solution {

public:

    TreeNode\* convertBST(TreeNode\* root) {

        int sum = 0;

        traverseBST(root, sum);

        return root;

    }

    void traverseBST(TreeNode\* root, int& sum) {

        if(!root) return;

        traverseBST(root->right, sum);

        root->val += sum;

        sum = root->val;

        traverseBST(root->left, sum);

    }

};

## 98 验证二叉搜索树

难度3

class Solution {

public:

    bool isValidBST(TreeNode\* root) {

        return isValidBST(root, nullptr, nullptr);

    }

    bool isValidBST(TreeNode\* root, TreeNode\* min, TreeNode\* max) {

        if(!root) return true;

        if(min != nullptr && root->val <= min->val) return false;

        if(max != nullptr && root->val >= max->val) return false;

        return isValidBST(root->left, min, root) &&

                isValidBST(root->right, root, max);

    }

};

# 动态规划

## 322 零钱兑换

class Solution {

public:

    int coinChange(vector<int>& coins, int amount) {

        vector<int> dp(amount+1, amount+1);

        dp[0] = 0;

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

            for(auto coin : coins) {

                if(i-coin < 0) continue;

                dp[i] = min(dp[i], dp[i-coin]+1);

            }

        }

        return (dp[amount] == amount+1) ? -1 : dp[amount];

    }

};

## 72 编辑距离

class Solution {

public:

    int minDistance(string word1, string word2) {

        int m = word1.length(), n = word2.length();

        vector<vector<int>> dp(m+1, vector<int>(n+1, 0));

        for(int i=0; i<=m; ++i) dp[i][0] = i;

        for(int j=0; j<=n; ++j) dp[0][j] = j;

        for(int i=1; i<=m; ++i) {

            for(int j=1; j<=n; ++j) {

                if(word1[i-1] == word2[j-1]) {   // 这里一定要注意！！

                    dp[i][j] = dp[i-1][j-1];

                }

                else {

                    dp[i][j] = min(min(dp[i-1][j-1], dp[i][j-1]), dp[i-1][j]) + 1;

                }

            }

        }

        return dp[m][n];

    }

};

/\*

状态，也就是决定问题规模的量是：i,j，即游标

选择，插入、删除、替换，或者啥也不做

定义，word1[0…i]，word2[0…j]的最小编辑距离：dp[i][j]

转移方程:

if(w1[i]==w2[j]) dp[i][j] = dp[i-1][j-1]

else dp[i][j] = min(选择) + 1

Base case ：dp[0][j] = j    dp[i][0] = i

返回值 dp[m][n]

\*/

## 1143 最长公共子序列

class Solution {

public:

    int longestCommonSubsequence(string text1, string text2) {

        int m = text1.length(), n = text2.length();

        vector<vector<int>> dp(m+1, vector<int>(n+1, 0));

        for(int i=1; i<=m; ++i) {

            for(int j=1; j<=n; ++j) {

                if(text1[i-1] == text2[j-1]) {

                    dp[i][j] = dp[i-1][j-1] + 1;

                }

                else {

                    dp[i][j] = max(dp[i-1][j], dp[i][j-1]);

                }

            }

        }

        return dp[m][n];

    }

};

## 516 最长回文子序列

class Solution {

public:

    int longestPalindromeSubseq(string s) {

        int n = s.length();

        vector<vector<int>> dp(n, vector<int>(n, 0));

        for(int i=0; i<n; ++i) dp[i][i] = 1;

        for(int i=n-1; i>=0; --i) {

            for(int j=i+1; j<n; ++j) {

                if(s[i] == s[j]) dp[i][j] = dp[i+1][j-1] + 2;

                else dp[i][j] = max(dp[i+1][j], dp[i][j-1]);

            }

        }

        return dp[0][n-1];

    }

};

/\*

采用双游标i，j

最优子结构，可以动态规划

状态：i，j

选择：s[i-1]是否等于s[j+1]

定义：dp[i][j]  i-j的长度

方程：等于，则加2；不等，则取个最大

接下来是难点：

Base case：i==j

遍历方向：反着或斜着遍历。见下页

\*/

拓展 ：

1. 能否压缩空间
2. 斜向遍历怎么写
3. 路径如何记录

class Solution {

public:

    int findTargetSumWays(vector<int>& nums, int S) {

        int sum = 0, method\_num = 0;

        backtrack(nums, 0, S, sum, method\_num);

        return method\_num;

    }

    void backtrack(vector<int>& nums, int idx, int S, int& sum, int& method\_num) {

        if(idx == nums.size()) {

            if(sum == S) {

                ++method\_num;

            }

            return;

        }

        sum += nums[idx];

        backtrack(nums, idx+1, S, sum, method\_num);

        sum -= nums[idx];

        sum -= nums[idx];

        backtrack(nums, idx+1, S, sum, method\_num);

        sum += nums[idx];

    }

};

## 300 最长递增子序列

标准解法：

class Solution {

public:

    int lengthOfLIS(vector<int>& nums) {

        int n = nums.size();

        int max\_length = 0;

        vector<int> dp(n,1);

        for(int i=0; i<n; ++i) {

            for(int j=0; j<i; ++j) {

                if(nums[i] > nums[j])

                    dp[i] = max(dp[i], dp[j]+1);

            }

            max\_length = max(max\_length, dp[i]);

        }

        return max\_length;

    }

};

class Solution {

public:

    int lengthOfLIS(vector<int>& nums) {

        int n = nums.size();

        if(n<=1) return n;

        vector<int> obj;

        for(int i=0; i<n; ++i) {

            if(obj.empty() || nums[i] > obj.back())

                obj.push\_back(nums[i]);

            else {

                auto iter = lower\_bound(obj.begin(), obj.end(), nums[i]);

                \*iter = nums[i];

            }

        }

        return obj.size();

    }

};

// 标准解法 - 二分查找 + 贪心法

class Solution {

public:

    int lengthOfLIS(vector<int>& nums) {

        int n = nums.size();

        if(n<=1) return n;

        vector<int> obj;

        for(int i=0; i<n; ++i) {

            if(obj.empty() || nums[i] > obj.back())

                obj.push\_back(nums[i]);

            else {

                int l=0, r=obj.size();

                while(l<r) {

                    int mid = l + (r-l)/2;

                    if(nums[i] > obj[mid])

                        l = mid+1;

                    else if (nums[i] < obj[mid])

                        r = mid;

                    else if (nums[i] == obj[mid])

                        r = mid;

                }

                obj[l] = nums[i];

            }

        }

        return obj.size();

    }

};

// 标准解法 – 自研二分查找 + 贪心法

class Solution {

public:

    int lengthOfLIS(vector<int>& nums) {

        int n = nums.size();

        if(n<=1) return n;

        int piles = 0;

        vector<int> top(n);   // 维护每个堆顶

        for(int i=0; i<n; ++i) {

            int poker = nums[i];

            int l=0, r=piles;

            while(l<r) {

                int mid = l + (r-l)/2;

                if(top[mid] > nums[i])

                    r = mid;

                if(top[mid] < nums[i])

                    l = mid + 1;

                if(top[mid] == nums[i])

                    r = mid;

            }

            if(l == piles) ++piles;

            top[l] = poker;

        }

        return piles;

    }

};

// 标准解法 - 二分查找 + 扑克堆

## 673 最长递增子序列的个数

class Solution {

public:

    int findNumberOfLIS(vector<int>& nums) {

        int n = nums.size();

        if(n<=1) return n;

        vector<int> dp\_length(n,1);

        vector<int> dp\_count(n,1);

        int max\_length = 0;

        for(int i=0; i<n; ++i) {

            for(int j=0; j<i; ++j) {

                if(nums[i] > nums[j]) {

                    if(dp\_length[j] >= dp\_length[i]) {

                        dp\_length[i] = dp\_length[j] + 1;

                        dp\_count[i] = dp\_count[j];

                    }

                    else if (dp\_length[j]+1 == dp\_length[i]) {

                        dp\_count[i] += dp\_count[j];

                    }

                }

            }

            if(dp\_length[i] > max\_length) {

                max\_length = dp\_length[i];

            }

        }

        int ret = 0;

        for(int i=0; i<n; ++i) {

            if(dp\_length[i] == max\_length) {

                ret += dp\_count[i];

            }

        }

        return ret;

    }

};

## 494 目标和

class Solution {

public:

    int findTargetSumWays(vector<int>& nums, int S) {

        int sum = accumulate(nums.begin(), nums.end(), 0);

        if(sum < S || (sum + S) % 2 == 1) return 0;

        return subset(nums, (sum + S)/2);

    }

    int subset(vector<int>& nums, int sum) {

        int n = nums.size();

        vector<vector<int>> dp(n+1, vector<int>(sum+1, 0));

        for(int i=0; i<=n; ++i)

            dp[i][0] = 1;

        for(int i=1; i<=n; ++i) {

            for(int j=0; j<=sum; ++j) {   // 细节，j从0开始

                if(j<nums[i-1]) {

                    dp[i][j] = dp[i-1][j];

                }

                else {

                    dp[i][j] = dp[i-1][j] + dp[i-1][j-nums[i-1]];

                }

            }

        }

        return dp[n][sum];

    }

};

压缩空间版本：

class Solution {

public:

    int findTargetSumWays(vector<int>& nums, int S) {

        int sum = accumulate(nums.begin(), nums.end(), 0);

        if(sum < S || (sum + S) % 2 == 1) return 0;

        return subset(nums, (sum + S)/2);

    }

    int subset(vector<int>& nums, int sum) {

        int n = nums.size();

        // vector<vector<int>> dp(n+1, vector<int>(sum+1, 0));

        vector<int> dp(sum+1, 0);

        dp[0] = 1;

        for(int i=1; i<=n; ++i) {

            for(int j=sum; j>=0; --j) {   // 细节，j从0开始

                if(j>=nums[i-1]) {

                    dp[j] = dp[j] + dp[j-nums[i-1]];

                }

            }

        }

        return dp[sum];

    }

};

// 压缩空间版本

## 416 分割等和子集

class Solution {

public:

    bool canPartition(vector<int>& nums) {

        int sum = accumulate(nums.begin(), nums.end(), 0);

        if(sum%2 == 1) return false;

        int n = nums.size();

        int target = sum/2;

        vector<vector<bool>> dp(n+1, vector<bool>(target + 1, false));

        for(int i=0; i<=n; ++i) {

            dp[i][0] = true;

        }

        for(int i=1; i<=n; ++i) {

            for(int j=0; j<=target; ++j) {

                if(j<nums[i-1]) {

                    dp[i][j] = dp[i-1][j];

                } else {

                    dp[i][j] = dp[i-1][j] || dp[i-1][j-nums[i-1]];

                }

            }

        }

        return dp[n][target];

    }

};

压缩空间 ：

class Solution {

public:

    bool canPartition(vector<int>& nums) {

        int sum = accumulate(nums.begin(), nums.end(), 0);

        if(sum%2 == 1) return false;

        int n = nums.size();

        int target = sum/2;

        vector<bool> dp(target + 1, false);

        dp[0] = true;

        for(int i=1; i<=n; ++i) {

            for(int j=target; j>=0; --j) {

                if(j>=nums[i-1]) {

                    dp[j] = dp[j] || dp[j-nums[i-1]];

                }

            }

        }

        return dp[target];

    }

};

## 887 鸡蛋掉落

class Solution {

public:

    int superEggDrop(int K, int N) {

        vector<vector<int>> memo(K+1, vector<int>(N+1, 0));

        return dp(K, N, memo);

    }

    int dp(int K, int N, vector<vector<int>>& memo) {

        if(K==1) return N;

        if(N==0) return 0;

        if(memo[K][N] != 0) return memo[K][N];

        int ret = INT\_MAX;

        // 原始版本

        // for(int i=1; i<=N; ++i) {

        //     int temp = max(dp(K-1, i-1, memo), dp(K, N-i, memo)) + 1;

        //     ret = min(ret, temp);

        // }

        // 二分版本

        int l=1, r=N;

        while(l<=r) {

            int mid = l + (r-l)/2;

            int broken = dp(K-1, mid-1, memo);

            int no\_broken = dp(K, N-mid, memo);

            if(broken < no\_broken) {

                l = mid+1;

                ret = min(ret, no\_broken+1);

            }

            else {

                r = mid-1;

                ret = min(ret, broken+1);

            }

        }

        memo[K][N] = ret;

        return ret;

    }

};

## 10 正则表达式匹配

暴力递归 ：

class Solution {

public:

    bool isMatch(string s, string p) {

        if(p.empty()) return s.empty();

        bool first\_match = !s.empty() && (s[0] == p[0] || p[0] == '.');

        if(p.length() >= 2 && p[1] == '\*') {

            return isMatch(s,  p.substr(2, p.length()-2)) ||

                (first\_match && isMatch(s.substr(1, s.length()-1), p));

        }

        else

            return first\_match && isMatch(s.substr(1, s.length()-1), p.substr(1, p.length()-1));

    }

};

备忘录版本：

class Solution {

public:

    bool isMatch(string s, string p) {

        int len\_s = s.length(), len\_p = p.length();

        vector<vector<int>> memo(len\_s+1, vector<int>(len\_p+1, -1));

        return isMatchHelper(s, p, 0, 0, memo);

    }

    bool isMatchHelper(const string& s, const string& p, int i, int j, vector<vector<int>>& memo) {

        if(memo[i][j] != -1) return (bool)memo[i][j];

        if(j == p.length()) return i == s.length();

        bool first\_match = i<s.length() && (s[i]==p[j] || p[j] == '.');

        if(j<=p.length()-2 && p[j+1]=='\*') {

            memo[i][j] = (int)(isMatchHelper(s, p, i, j+2, memo) ||

                               (first\_match && isMatchHelper(s, p, i+1, j, memo)));

        }

        else {

            memo[i][j] = (int)(first\_match && isMatchHelper(s, p, i+1, j+1, memo));

        }

        return (bool)memo[i][j];

    }

};

## 312 戳气球

class Solution {

public:

    int maxCoins(vector<int>& nums) {

        int n = nums.size();

        vector<int> points(n+2);

        points[0] = 1;

        points[n+1] = 1;

        for(int i=1; i<=n; ++i) {

            points[i] = nums[i-1];

        }

        vector<vector<int>> dp(n+2, vector<int>(n+2, 0));

        for(int i=n; i>=0; --i) {

            for(int j=i+1; j<n+2; ++j) {

                for(int k=i+1; k<j; ++k) {

                    dp[i][j] = max(dp[i][j],

                        dp[i][k] + dp[k][j] + points[i]\*points[j]\*points[k]);

                }

            }

        }

        return dp[0][n+1];

    }

};

## 877 石头游戏

动态规划解法：

class Solution {

public:

    typedef pair<int, int> Tp;

    bool stoneGame(vector<int>& piles) {

        int n = piles.size();

        vector<vector<Tp>> dp(n, vector<Tp>(n, {0, 0}));

        for(int i=0; i<n; ++i) {

            dp[i][i].first = piles[i];

            dp[i][i].second = 0;

        }

        // 方程为j-i = k (k=1,2...n-1)

        for(int k=1; k<=n-1; ++k) {

            for(int i=0; i<=n-1-k; ++i) {

                int j = i+k;

                int left = piles[i] + dp[i+1][j].second;

                int right = piles[j] + dp[i][j-1].second;

                if(left > right) {

                    dp[i][j].first = left;

                    dp[i][j].second = dp[i+1][j].first;

                }

                else {

                    dp[i][j].first = right;

                    dp[i][j].second = dp[i][j-1].first;

                }

            }

        }

        return dp[0][n-1].first > dp[0][n-1].second;

    }

};

智力题解法 ： return true;

## 650 只有两个键的键盘

class Solution {

public:

    int minSteps(int n) {

        vector<int> dp(n+1);

        for(int i=2; i<=n; ++i) {

            dp[i] = i;

            for(int j=2; j<=i/2; ++j) {

                if(i%j == 0) {

                    dp[i] = dp[j] + dp[i/j];

                    break;

                }

            }

        }

        return dp[n];

    }

};

## 651 4键键盘

class Solution {

public:

    int maxA(int N) {

        vector<int> dp(N+1);

        dp[0] = 0;

        for(int i=1; i<=N; ++i) {

            dp[i] = i;

            for(int j=2; j<i; ++j) {

                dp[i] = max(dp[i], (i-j+1)\*dp[j-2]);

            }

        }

        return dp[N];

    }

};

## 1312 让字符串成为回文串的最少插入次数

class Solution {

public:

    int minInsertions(string s) {

        int n = s.length();

        vector<vector<int>> dp(n, vector<int>(n,0));

        for(int i=n-2; i>=0; i--) {

            for(int j=i+1; j<n; ++j) {

                if(s[i] == s[j]) {

                    dp[i][j] = dp[i+1][j-1];

                }

                else {

                    dp[i][j] = min(dp[i+1][j], dp[i][j-1]) + 1;

                }

            }

        }

        return dp[0][n-1];

    }

};

## 354 俄罗斯套娃信封问题

俄罗斯套娃信封问题

class Solution {public:

int maxEnvelopes(vector<vector<int>>& envelopes) {

sort(envelopes.begin(), envelopes.end(),

[](const vector<int>& a, const vector<int>& b) {

if(a[0] == b[0]) return a[1] > b[1];

return a[0] < b[0];} );

int n = envelopes.size();

vector<int> lis;

for(int i=0; i<n; ++i) {

if(lis.empty() || lis.back() < envelopes[i][1]) {

lis.push\_back(envelopes[i][1]);

}

else {

int l=0, r=lis.size();

while(l<r) {

int mid = l + (r-l)/2;

if(envelopes[i][1] > lis[mid]) {

l = mid + 1;

}

else if(envelopes[i][1] < lis[mid]) {

r = mid;

}

else if(envelopes[i][1] == lis[mid]) {

r = mid;

}

}

lis[l] = envelopes[i][1];

}

}

return lis.size();

}

bool compare(const vector<int>& a, const vector<int>& b) {

if(a[0] == b[0]) return a[1] > b[1];

return a[0] < b[0];

}

};

## 53 最大子序和

class Solution {

public:

    int maxSubArray(vector<int>& nums) {

        int n = nums.size();

        vector<int> dp(n, INT\_MIN);

        dp[0] = nums[0];

        int ret = dp[0];

        for(int i=1; i<n; ++i) {

            dp[i] = max(nums[i], dp[i-1] + nums[i]);

            ret = max(ret, dp[i]);

        }

        return ret;

    }

};

## 198 打家劫舍

class Solution {

public:

    int rob(vector<int>& nums) {

        int n = nums.size();

        vector<int> dp(n+2, 0);

        for(int i=n-1; i>=0; --i) {

            dp[i] = max(dp[i+1], dp[i+2] + nums[i]);

        }

        return dp[0];

    }

};

降低空间复杂度，将一维数组变成几个变量，相当于滑块操作

class Solution {

public:

    int rob(vector<int>& nums) {

        int n = nums.size();

        int score = 0, score\_1 = 0, score\_2 = 0;

        for(int i=n-1; i>=0; --i) {

            score = max(score\_1, score\_2 + nums[i]);

            score\_2 = score\_1;

            score\_1 = score;

        }

        return score;

    }

};

## 213 打家劫舍 II

class Solution {

public:

    int rob(vector<int>& nums) {

        int n = nums.size();

        if(n==1) return nums[0];

        return max(rob(nums, 0, n-2), rob(nums, 1, n-1));

    }

    int rob(vector<int>& nums, int start, int end) {

        int score = 0, score\_1 = 0, score\_2 = 0;

        for(int i=end; i>=start; --i) {

            score = max(score\_1, nums[i]+score\_2);

            score\_2 = score\_1;

            score\_1 = score;

        }

        return score;

    }

};

## 337 打家劫舍 III

/\*\*

 \* Definition for a binary tree node.

 \* struct TreeNode {

 \*     int val;

 \*     TreeNode \*left;

 \*     TreeNode \*right;

 \*     TreeNode(int x) : val(x), left(NULL), right(NULL) {}

 \* };

 \*/

class Solution {

public:

    int rob(TreeNode\* root) {

        pair<int,int> raw = maxRob(root);

        return max(raw.first, raw.second);

    }

    pair<int, int> maxRob(TreeNode\* root) {

        if(!root) return {0,0};

        pair<int,int> left = maxRob(root->left);

        pair<int,int> right = maxRob(root->right);

        int rob\_root = root->val + left.second + right.second;

        int norob\_root = max(left.first, left.second) + max(right.first, right.second);

        return {rob\_root, norob\_root};

    }

};

## 121 买卖股票的最佳时机

class Solution {

public:

    int maxProfit(vector<int>& prices) {

        int n = prices.size();

        vector<vector<int>> dp(n+1, vector<int>(2,0));

        dp[0][0] = 0;

        dp[0][1] = INT\_MIN;

        for(int i=1; i<=n; ++i) {

            dp[i][0] = max(dp[i-1][0], dp[i-1][1] + prices[i-1]);

            dp[i][1] = max(dp[i-1][1],  -prices[i-1]);

        }

        return dp[n][0];

    }

};

压缩状态：

class Solution {

public:

    int maxProfit(vector<int>& prices) {

        int n = prices.size();

        vector<vector<int>> dp(n+1, vector<int>(2,0));

        int dp\_i\_0 = 0, dp\_i\_1 = INT\_MIN;

        for(int i=1; i<=n; ++i) {

            dp\_i\_0 = max(dp\_i\_0, dp\_i\_1 + prices[i-1]);

            dp\_i\_1 = max(dp\_i\_1,  -prices[i-1]);

        }

        return dp\_i\_0;

    }

};

## 122 买卖股票的最佳时机 II

class Solution {

public:

    int maxProfit(vector<int>& prices) {

        int n = prices.size();

        int dp\_i\_0 = 0, dp\_i\_1 = INT\_MIN;

        for(int i=1; i<=n; ++i) {

            dp\_i\_0 = max(dp\_i\_0, dp\_i\_1 + prices[i-1]);

            dp\_i\_1 = max(dp\_i\_1, dp\_i\_0 - prices[i-1]);

        }

        return dp\_i\_0;

    }

};

## 309 最佳买卖股票时机含冷冻期

class Solution {

public:

    int maxProfit(vector<int>& prices) {

        int n = prices.size();

        int dp\_i\_0 = 0, dp\_i\_1 = INT\_MIN;

        int pre = 0;

        for(int i=1; i<=n; ++i) {

            int temp = dp\_i\_0;

            dp\_i\_0 = max(dp\_i\_0, dp\_i\_1 + prices[i-1]);

            dp\_i\_1 = max(dp\_i\_1, pre - prices[i-1]);

            pre = temp;

        }

        return dp\_i\_0;

    }

};

## 714 买卖股票的最佳时机含手续费

class Solution {

public:

    int maxProfit(vector<int>& prices, int fee) {

        int n = prices.size();

        int dp\_i\_0 = 0, dp\_i\_1 = INT\_MIN;

        for(int i=1; i<=n; ++i) {

            dp\_i\_0 = max(dp\_i\_0, dp\_i\_1 + prices[i-1]);

            dp\_i\_1 = max(dp\_i\_1, dp\_i\_0 - prices[i-1] - fee);

        }

        return dp\_i\_0;

    }

};

## 123 买卖股票的最佳时机 III

class Solution {

public:

    int maxProfit(vector<int>& prices) {

        int n = prices.size();

        vector<vector<vector<int>>> dp(n+1,

            vector<vector<int>>(3, vector<int>(2, 0)));

        // dp[0][k][0] = 0

        // dp[0][k][1] = -infinity

        // dp[i][0][0] = 0

        // dp[i][0][1] = -infinity

        for(int ik = 1; ik<=2; ++ik) dp[0][ik][1] = INT\_MIN;

        for(int ii = 1; ii<=n; ++ii) dp[ii][0][1] = INT\_MIN;

        for(int i=1; i<=n; ++i) {

            for(int k=1; k<=2; ++k) {

                dp[i][k][0] = max(dp[i-1][k][0], dp[i-1][k][1] + prices[i-1]);

                dp[i][k][1] = max(dp[i-1][k][1], dp[i-1][k-1][0] - prices[i-1]);

            }

        }

        return dp[n][2][0];

    }

};

## 188 买卖股票的最佳时机 IV

class Solution {

public:

    int maxProfit(int k, vector<int>& prices) {

        int n = prices.size();

        if( n/2 < k) {

            vector<vector<int>> dp(n+1, vector<int>(2,0));

            dp[0][1] = INT\_MIN;

            for(int i=1; i<=n; ++i) {

                dp[i][0] = max(dp[i-1][0], dp[i-1][1] + prices[i-1]);

                dp[i][1] = max(dp[i-1][1], dp[i-1][0] - prices[i-1]);

            }

            return dp[n][0];

        }

        else {

            vector<vector<vector<int>>> dp(n+1,

                vector<vector<int>>(k+1, vector<int>(2, 0)));

            for(int ik = 1; ik<=k; ++ik) dp[0][ik][1] = INT\_MIN;

            for(int ii = 1; ii<=k; ++ii) dp[ii][0][1] = INT\_MIN;

            for(int i=1; i<=n; ++i) {

                for(int kk=1; kk<=k; ++kk) {

                    dp[i][kk][0] = max(dp[i-1][kk][0],dp[i-1][kk][1] + prices[i-1]);

                    dp[i][kk][1] = max(dp[i-1][kk][1],dp[i-1][kk-1][0] - prices[i-1]);

                }

            }

            return dp[n][k][0];

        }

    }

};

# 智力题

## 292 Nim 游戏

class Solution {

public:

    bool canWinNim(int n) {

        return n%4;

    }

};

## 319 灯泡开关

return (int)sqrt(n);

# 回溯法

## 46 全排列

Contain法：

class Solution {

public:

    vector<vector<int>> permute(vector<int>& nums) {

        vector<int> track;

        vector<vector<int>> ans;

        backtrack(nums, track, ans);

        return ans;

}

    void backtrack(vector<int>& nums, vector<int>& track, vector<vector<int>>& ans) {

        if(track.size() == nums.size()) {

            ans.push\_back(track);

        }

        for(int i=0; i<nums.size(); ++i) {

            if(find(track.begin(), track.end(), nums[i]) != track.end()) continue;

            track.push\_back(nums[i]);

            backtrack(nums, track, ans);

            track.pop\_back();

        }

    }

};

Swap 法：

class Solution {

public:

    vector<vector<int>> permute(vector<int>& nums) {

        vector<int> track;

        vector<vector<int>> ans;

        backtrack(nums, 0, track, ans);

        return ans;

    }

    void backtrack(vector<int>& nums, int start, vector<int>& track, vector<vector<int>>& ans) {

        if(start == nums.size()) {

            ans.push\_back(track);

            return;

        }

        for(int i=start; i<nums.size(); ++i) {

            swap(nums[start], nums[i]);

            track.push\_back(nums[start]);

            backtrack(nums, start+1, track, ans);

            track.pop\_back();

            swap(nums[start], nums[i]);

        }

    }

};

## 47 全排列II

增加一个同层去重，搞定

class Solution {

public:

    vector<vector<int>> permuteUnique(vector<int>& nums) {

        vector<int> track;

        vector<vector<int>> ans;

        backtrack(nums, 0, track, ans);

        return ans;

    }

    void backtrack(vector<int>& nums, int start, vector<int>& track, vector<vector<int>>& ans) {

        if(start == nums.size()) {

            ans.push\_back(track);

            return;

        }

        unordered\_set<int> memo;

        for(int i=start; i<nums.size(); ++i) {

            if(memo.count(nums[i])) continue;

            memo.insert(nums[i]);

            swap(nums[start], nums[i]);

            track.push\_back(nums[start]);

            backtrack(nums, start+1, track, ans);

            track.pop\_back();

            swap(nums[start], nums[i]);

        }

    }

};

## 78 子集

class Solution {

public:

    vector<vector<int>> subsets(vector<int>& nums) {

        vector<int> track;

        vector<vector<int>> ans;

        backtrack(nums, 0, track, ans);

        return ans;

    }

    void backtrack(vector<int>& nums, int start, vector<int>& track, vector<vector<int>>& ans) {

        ans.push\_back(track);

        // if(track.size() == nums.size()) return;

        for(int i=start; i<nums.size(); ++i) {

            track.push\_back(nums[i]);

            backtrack(nums, i+1, track, ans);

            track.pop\_back();

        }

    }

};

## 90 子集II

class Solution {

public:

    vector<vector<int>> subsetsWithDup(vector<int>& nums) {

        vector<int> track;

        vector<vector<int>> ans;

        sort(nums.begin(), nums.end());

        backtrack(nums, 0, track, ans);

        return ans;

    }

    void backtrack(vector<int>& nums, int start, vector<int>& track, vector<vector<int>>& ans) {

        ans.push\_back(track);

        unordered\_set<int> hash;

        for(int i=start; i<nums.size(); ++i) {

            if(hash.count(nums[i])) continue;

            track.push\_back(nums[i]);

            hash.insert(nums[i]);

            backtrack(nums, i+1, track, ans);

            track.pop\_back();

        }

    }

};

## 77 组合

class Solution {

public:

    vector<vector<int>> combine(int n, int k) {

        vector<int> track;

        vector<vector<int>> ans;

        backtrack(track, ans, 1, n, k);

        return ans;

    }

    void backtrack(vector<int>& track, vector<vector<int>>& ans, int start, int n, int k) {

        if(track.size() == k) {

            ans.push\_back(track);

            return;

        }

        for(int i=start; i<=n; ++i) {

            track.push\_back(i);

            backtrack(track, ans, i+1, n, k);

            track.pop\_back();

        }

    }

};

## 51 N皇后

class Solution {

public:

    vector<vector<string>> solveNQueens(int n) {

        vector<string> board(n, string(n, '.'));

        vector<vector<string>> ans;

        backtrack(board, 0, ans);

        return ans;

    }

    void backtrack(vector<string>& board, int row, vector<vector<string>>& ans) {

        int n = board.size();

        if (row == n) {

            ans.push\_back(board);

            return;

        }

        for(int col=0; col<n; ++col) {

            if(!isValid(board, row, col)) continue;

            board[row][col] = 'Q';

            backtrack(board, row+1, ans);

            board[row][col] = '.';

        }

    }

    bool isValid(vector<string>& board, int row, int col) {

        int n = board.size();

        for(int i=0; i<row; ++i) {

            if(board[i][col] == 'Q') return false;

        }

        for(int i=row-1, j=col-1; i>=0 && j>=0; --i, --j) {

            if(board[i][j] == 'Q') return false;

        }

        for(int i=row-1, j=col+1; i>=0 && j<n; --i, ++j) {

            if(board[i][j] == 'Q') return false;

        }

        return true;

    }

};

## 52 N皇后II

class Solution {

public:

    int totalNQueens(int n) {

        vector<string> board(n, string(n, '.'));

        int ans\_num = 0;

        backtrack(board, 0, ans\_num);

        return ans\_num;

    }

    void backtrack(vector<string>& board, int row, int& ans\_num) {

        int n = board.size();

        if(row == n) {

            ++ans\_num;

            return;

        }

        for(int col=0; col<n; ++col) {

            if(!isValid(board, row, col)) continue;

            board[row][col] = 'Q';

            backtrack(board, row+1, ans\_num);

            board[row][col] = '.';

        }

    }

    bool isValid(vector<string>& board, int row, int col) {

        int n = board.size();

        for(int i=0; i<row; ++i) {

            if(board[i][col] == 'Q') return false;

        }

        for(int i=row-1, j=col-1; i>=0 && j>=0; --i, --j) {

            if(board[i][j] == 'Q') return false;

        }

        for(int i=row-1, j=col+1; i>=0 && j<n; --i, ++j) {

            if(board[i][j] == 'Q') return false;

        }

        return true;

    }

};

## 22 括号生成

class Solution {

public:

    vector<string> generateParenthesis(int n) {

        string track;

        vector<string> ans;

        backtrack(n, n, track, ans);

        return ans;

    }

    void backtrack(int left, int right, string& track, vector<string>& ans) {

        if(left == -1 || right == -1) return;

        if(left > right) return;

        if(left == 0 && right == 0) {

            ans.push\_back(track);

            return;

        }

        track.push\_back('(');

        backtrack(left-1, right, track, ans);

        track.pop\_back();

        track.push\_back(')');

        backtrack(left, right-1, track, ans);

        track.pop\_back();

    }

};

## 36 有效的数独

class Solution {

public:

    bool isValidSudoku(vector<vector<char>>& board) {

        vector<vector<bool>> a(9, vector<bool>(9, false));

        vector<vector<bool>> b(9, vector<bool>(9, false));

        vector<vector<bool>> c(9, vector<bool>(9, false));

        for(int i=0; i<9; ++i) {

            for(int j=0; j<9; ++j) {

                if(board[i][j] == '.') continue;

                int k = board[i][j] - '1';

                int p = (i/3)\*3 + (j/3);

                if(a[i][k] || b[j][k] || c[p][k]) return false;

                a[i][k] = true;

                b[j][k] = true;

                c[p][k] = true;

            }

        }

        return true;

    }

};

## 37 解数独

class Solution {

public:

    void solveSudoku(vector<vector<char>>& board) {

        backtrack(board, 0, 0);

    }

    bool backtrack(vector<vector<char>>& board, int r, int c) {

        int m = board.size();

        int n = board[0].size();

        if(c==n) return backtrack(board, r+1, 0);

        if(r==m) {

            return true;

        }

        for(int i=r; i<m; ++i) {

            for(int j=c; j<n; ++j) {

                if(board[i][j] != '.') {

                    return backtrack(board, i, j+1);

                }

                for(char ch='1'; ch<='9'; ++ch) {

                    if(!isValid(board, i, j, ch)) continue;

                    board[i][j] = ch;

                    if(backtrack(board, i, j+1)) return true;

                    board[i][j] = '.';

                }

                return false;

            }

        }

        return false;

    }

    bool isValid(vector<vector<char>>& board, int r, int c, char ch) {

        for(int i=0; i<9; ++i) {

            if(board[i][c] == ch) return false;

            if(board[r][i] == ch) return false;

            int m = (r/3)\*3 + i/3;

            int n = (c/3)\*3 + i%3;

            if(board[m][n] == ch) return false;

        }

        return true;

    }

};

# 设计类

## 146 LRU缓存机制

struct Node {

    int key;

    int val;

    Node \*next, \*prev;

    Node(int k, int v) {

        key = k;

        val = v;

        next = nullptr;

        prev = nullptr;

    }

    ~Node() {

        if(next) {

            delete next;

            next = nullptr;

        }

        if(prev) {

            delete prev;

            prev = nullptr;

        }

    }

};

class DoubleList {

private:

    Node \*head\_, \*tail\_;

    int size\_;

public:

    // 初始化

    DoubleList() {

        head\_ = new Node(0, 0);

        tail\_ = new Node(0, 0);

        head\_->next = tail\_;

        tail\_->prev = head\_;

        size\_ = 0;

    }

    // 在尾部添加节点x

    void addLast(Node\* x) {

        Node\* rear = tail\_->prev;

        rear->next = x;

        x->prev = rear;

        x->next = tail\_;

        tail\_->prev = x;

        ++size\_;

    }

    // 删除节点x

    void remove(Node\* x) {

        x->prev->next = x->next;

        x->next->prev = x->prev;

        x->prev = nullptr;

        x->next = nullptr;

        --size\_;

    }

    // 删除第一个节点，并返回

    Node\* removeFirst() {

        if(head\_->next == tail\_) return nullptr;

        Node\* first = head\_->next;

        remove(first);

        return first;

    }

    int size() const { return size\_; }

};

class LRUCache {

private:

    unordered\_map<int, Node\*> map\_;

    DoubleList cache\_;

    int cap\_;

    /\* 将某个 key 提升为最近使用的 \*/

    void makeRecently(int key) {

        Node\* x = map\_[key];

        cache\_.remove(x);

        cache\_.addLast(x);

    }

    /\* 添加最近使用的元素 \*/

    void addRecently(int key, int val) {

        Node\* x = new Node(key, val);

        cache\_.addLast(x);

        map\_[key] = x;

    }

    /\* 删除某一个 key \*/

    void deleteKey(int key) {

        Node\* x = map\_[key];

        cache\_.remove(x);

        map\_.erase(key);

    }

    /\* 删除最久未使用的元素 \*/

    void removeLeastRecently() {

        Node\* deleted\_node = cache\_.removeFirst();

        map\_.erase(deleted\_node->key);

    }

public:

    LRUCache(int capacity) {

        cap\_ = capacity;

    }

    int get(int key) {

        if(!map\_.count(key)) return -1;

        makeRecently(key);

        return map\_[key]->val;

    }

    void put(int key, int value) {

        if(map\_.count(key)) {

            map\_[key]->val = value;

            makeRecently(key);

        }

        else {

            if(cache\_.size() == cap\_) {

                removeLeastRecently();

            }

            addRecently(key, value);

        }

    }

};

/\*\*

 \* Your LRUCache object will be instantiated and called as such:

 \* LRUCache\* obj = new LRUCache(capacity);

 \* int param\_1 = obj->get(key);

 \* obj->put(key,value);

 \*/

## 460 LFU

class LinkedHashSet {

private:

    unordered\_map<int, list<int>::iterator> hash\_;

    list<int> list\_;

public:

    bool isEmpty() const {

        return list\_.empty() && hash\_.empty();

    }

    void addLast(int key) {

        if(hash\_.count(key)) {

            remove(key);

        }

        list\_.push\_back(key);

        auto it = list\_.end();

        hash\_[key] = --it;

    }

    void remove(int key) {

        list<int>::iterator it = hash\_[key];

        list\_.erase(it);

        hash\_.erase(key);

    }

    int removeFirst() {

        auto first\_key = list\_.front();

        remove(first\_key);

        return first\_key;

    }

};

class LFUCache {

private:

    unordered\_map<int, int> key\_to\_val\_;

    unordered\_map<int, int> key\_to\_freq\_;

    unordered\_map<int, LinkedHashSet\*> freq\_to\_key\_;

    int cap\_;

    int min\_freq\_;

    void increaseFreq(int key) {

        int freq = key\_to\_freq\_[key];

        ++key\_to\_freq\_[key];

        freq\_to\_key\_[freq]->remove(key);

        if(!freq\_to\_key\_.count(freq+1)) {

            freq\_to\_key\_[freq+1] = new LinkedHashSet;

        }

        freq\_to\_key\_[freq+1]->addLast(key);

        if(freq\_to\_key\_[freq]->isEmpty()) {

            freq\_to\_key\_.erase(freq);

            if(freq == min\_freq\_) {

                ++min\_freq\_;

            }

        }

    }

    void removeMinFreqKey() {

        LinkedHashSet\* min\_freq\_key\_hash = freq\_to\_key\_[min\_freq\_];

        int deleted\_key = min\_freq\_key\_hash->removeFirst();

        if(min\_freq\_key\_hash->isEmpty()) {

            freq\_to\_key\_.erase(min\_freq\_);

        }

        key\_to\_val\_.erase(deleted\_key);

        key\_to\_freq\_.erase(deleted\_key);

    }

public:

    LFUCache(int capacity) {

        cap\_ = capacity;

        min\_freq\_ = 0;

    }

    int get(int key) {

        if(!key\_to\_val\_.count(key))

            return -1;

        increaseFreq(key);

        return key\_to\_val\_[key];

    }

    void put(int key, int value) {

        if(cap\_<=0) return;

        if(key\_to\_val\_.count(key)) {

            key\_to\_val\_[key] = value;

            increaseFreq(key);

        }

        else {

            if(key\_to\_val\_.size() == cap\_) {

                removeMinFreqKey();

            }

            key\_to\_val\_[key] = value;

            key\_to\_freq\_[key] = 1;

            if(!freq\_to\_key\_.count(1)) {

                freq\_to\_key\_[1] = new LinkedHashSet;

            }

            freq\_to\_key\_[1]->addLast(key);

            min\_freq\_ = 1;

        }

    }

};

## 335 设计推特

/\*\*

 \* Your LFUCache object will be instantiated and called as such:

 \* LFUCache\* obj = new LFUCache(capacity);

 \* int param\_1 = obj->get(key);

 \* obj->put(key,value);

 \*/

class Twitter {

private:

    static int timestamp;

    class Tweet{

    public:

        int id\_;

        int time\_;

        Tweet\* next\_;

        Tweet(int id, int time) : id\_(id), time\_(time), next\_(nullptr) {}

        struct Comp {

            bool operator() (Tweet\* t1, Tweet\* t2) {

                return t1->time\_ < t2->time\_;

            }

        };

    };

    class User{

    private:

        int id\_;

    public:

        unordered\_set<int> followed;

        Tweet\* head;

        User(int user\_id) {

            id\_ = user\_id;

            head = nullptr;

            // 自己关注自己

            followed.insert(user\_id);

        }

        // 关注

        void follow(int user\_id) {

            followed.insert(user\_id);

        }

        // 取关

        void unfollow(int user\_id) {

            if(user\_id != id\_)

                followed.erase(user\_id);

        }

        // 发推

        void post(int tweet\_id) {

            Tweet\* twt = new Tweet(tweet\_id, timestamp);

            ++timestamp;

            twt->next\_ = head;

            head = twt;

        }

    };

    unordered\_map<int, User\*> user\_map\_;

public:

    /\*\* Initialize your data structure here. \*/

    Twitter() { }

    /\*\* Compose a new tweet. \*/

    void postTweet(int userId, int tweetId) {

        if(!user\_map\_.count(userId)) {

            user\_map\_[userId] = new User(userId);

        }

        user\_map\_[userId]->post(tweetId);

    }

    /\*\* Retrieve the 10 most recent tweet ids in the user's news feed. Each item in the news feed must be posted by users who the user followed or by the user herself. Tweets must be ordered from most recent to least recent. \*/

    vector<int> getNewsFeed(int userId) {

        vector<int> res;

        if(!user\_map\_.count(userId)) return res;

        auto& followed = user\_map\_[userId]->followed;

        priority\_queue<Tweet\*, vector<Tweet\*>, Tweet::Comp> pq;

        for(int id : followed) {

            if(!user\_map\_.count(id)) continue;

            Tweet\* twt = user\_map\_[id]->head;

            if(twt != nullptr) {

                pq.push(twt);

            }

        }

        while(!pq.empty()) {

            if(res.size() == 10) break;

            Tweet\* twt = pq.top();

            res.push\_back(twt->id\_);

            pq.pop();

            if(twt->next\_ != nullptr) {

                pq.push(twt->next\_);

            }

        }

        return res;

    }

    /\*\* Follower follows a followee. If the operation is invalid, it should be a no-op. \*/

    void follow(int followerId, int followeeId) {

        if(!user\_map\_.count(followerId)) {

            user\_map\_[followerId] = new User(followerId);

        }

        if(!user\_map\_.count(followeeId)) {

            user\_map\_[followeeId] = new User(followeeId);

        }

        user\_map\_[followerId]->follow(followeeId);

    }

    /\*\* Follower unfollows a followee. If the operation is invalid, it should be a no-op. \*/

    void unfollow(int followerId, int followeeId) {

        if(user\_map\_.count(followerId)) {

            user\_map\_[followerId]->unfollow(followeeId);

        }

    }

};

int Twitter::timestamp = 0;

/\*\*

 \* Your Twitter object will be instantiated and called as such:

 \* Twitter\* obj = new Twitter();

 \* obj->postTweet(userId,tweetId);

 \* vector<int> param\_2 = obj->getNewsFeed(userId);

 \* obj->follow(followerId,followeeId);

 \* obj->unfollow(followerId,followeeId);

 \*/

# 统计与随机

## 382 链表随机结点

/\*\*

 \* Definition for singly-linked list.

 \* struct ListNode {

 \*     int val;

 \*     ListNode \*next;

 \*     ListNode() : val(0), next(nullptr) {}

 \*     ListNode(int x) : val(x), next(nullptr) {}

 \*     ListNode(int x, ListNode \*next) : val(x), next(next) {}

 \* };

 \*/

class Solution {

private:

    ListNode\* head\_;

    default\_random\_engine e\_;

public:

    /\*\* @param head The linked list's head.

        Note that the head is guaranteed to be not null, so it contains at least one node. \*/

    Solution(ListNode\* head) {

        head\_ = head;

    }

    /\*\* Returns a random node's value. \*/

    int getRandom() {

        int res = 0;

        int i = 0;

        ListNode\* p = head\_;

        while(p != NULL) {

            uniform\_int\_distribution<int> u(0, i++);

            if( u(e\_) == 0 ) {

                res = p->val;

            }

            p = p->next;

        }

        return res;

    }

};

/\*\*

 \* Your Solution object will be instantiated and called as such:

 \* Solution\* obj = new Solution(head);

 \* int param\_1 = obj->getRandom();

 \*/

## 398 随机数索引

class Solution {

    vector<int>\* p\_nums\_;

public:

    Solution(vector<int>& nums) {

        p\_nums\_ = &nums;

    }

    int pick(int target) {

        int cnt = 0, ret = 0;

        for(int i=0; i<p\_nums\_->size(); ++i) {

            if(p\_nums\_->at(i) == target) {

                if(rand()%(++cnt) == 0) {

                    ret = i;

                }

            }

        }

        return ret;

    }

};

/\*\*

 \* Your Solution object will be instantiated and called as such:

 \* Solution\* obj = new Solution(nums);

 \* int param\_1 = obj->pick(target);

 \*/

## 528 依概率采样

class Solution {

private:

    vector<int> sums;

public:

    Solution(vector<int>& w) : sums(w) {

        partial\_sum(sums.begin(), sums.end(), sums.begin());

    }

    int pickIndex() {

        return lower\_bound(sums.begin(), sums.end(), rand()%sums.back() + 1) - sums.begin();

    }

};

/\*\*

 \* Your Solution object will be instantiated and called as such:

 \* Solution\* obj = new Solution(w);

 \* int param\_1 = obj->pickIndex();

 \*/

# 优先级队列

## 295 数据流的中位数

struct Comp {

    bool operator() (int a, int b) { return a > b;}

};

class MedianFinder {

public:

    /\*\* initialize your data structure here. \*/

    priority\_queue<int, vector<int>, Comp> large;

    priority\_queue<int, vector<int>> small;

    MedianFinder() {

}

    void addNum(int num) {

        if(large.size() >= small.size()) {

            large.push(num);

            small.push(large.top());

            large.pop();

        }

        else {

            small.push(num);

            large.push(small.top());

            small.pop();

        }

    }

    double findMedian() {

        if(large.size() == small.size()) {

            return (large.top() + small.top()) / 2.0;

        }

        else {

            return large.size() > small.size() ? large.top() : small.top();

        }

    }

};

/\*\*

 \* Your MedianFinder object will be instantiated and called as such:

 \* MedianFinder\* obj = new MedianFinder();

 \* obj->addNum(num);

 \* double param\_2 = obj->findMedian();

 \*/

## 335 设计推特（见 设计类）

# 单调栈

## 496 下一个更大元素 I

class Solution {

public:

    vector<int> nextGreaterElement(vector<int>& nums1, vector<int>& nums2) {

        vector<int> ans;

        stack<int> s;

        unordered\_map<int, int> hash\_map;

        for(int j=nums2.size()-1; j>=0; --j) {

            while(!s.empty() && s.top() <= nums2[j]) {

                s.pop();

            }

            hash\_map[nums2[j]] = s.empty() ? -1 : s.top();

            s.push(nums2[j]);

        }

        for(int i = 0; i<nums1.size(); ++i) {

            ans.push\_back(hash\_map[nums1[i]]);

        }

        return ans;

    }

};

## 503 下一个更大元素 II

class Solution {

public:

    vector<int> nextGreaterElements(vector<int>& nums) {

        int n = nums.size();

        vector<int> ans(n, 0);

        stack<int> s;

        for(int i=2\*n-1; i>=0; --i) {

            while(!s.empty() && nums[i%n] >= s.top()) {

                s.pop();

            }

            ans[i%n] = s.empty() ? -1 : s.top();

            s.push(nums[i%n]);

        }

        return ans;

    }

};

## 739 每日温度

class Solution {

public:

    vector<int> dailyTemperatures(vector<int>& T) {

        int n = T.size();

        stack<int> s;

        vector<int> ans(n, 0);

        for(int i=n-1; i>=0; --i) {

            while(!s.empty() && T[i] >= T[s.top()]) {

                s.pop();

            }

            ans[i] = s.empty() ? 0 : s.top() - i;

            s.push(i);

        }

        return ans;

    }

};

## 1019 链表中的下一个更大节点

/\*\*

 \* Definition for singly-linked list.

 \* struct ListNode {

 \*     int val;

 \*     ListNode \*next;

 \*     ListNode(int x) : val(x), next(NULL) {}

 \* };

 \*/

class Solution {

public:

    vector<int> nextLargerNodes(ListNode\* head) {

        vector<int> ans;

        int size = 0;

        stack<int> s;

        traverse(head, s, ans, size);

        return ans;

    }

    void traverse(ListNode\* head, stack<int>& s, vector<int>& ans, int& size) {

        if(!head) {

            ans.assign(size, 0);

            return;

        }

        ++size;

        traverse(head->next, s, ans, size);

        while(!s.empty() && head->val >= s.top()) {

            s.pop();

        }

        ans[--size] = s.empty() ? 0 : s.top();

        s.push(head->val);

        // 后序遍历代码

    }

};

# 栈和队列

## 232 用栈实现队列

class MyQueue {

private:

    stack<int> s\_in, s\_out;

    void move() {

        while(!s\_in.empty()) {

            s\_out.push(s\_in.top());

            s\_in.pop();

        }

    }

public:

    /\*\* Initialize your data structure here. \*/

    MyQueue() {

    }

    /\*\* Push element x to the back of queue. \*/

    void push(int x) {

        s\_in.push(x);

    }

    /\*\* Removes the element from in front of queue and returns that element. \*/

    int pop() {

        int ret = peek();

        s\_out.pop();

        return ret;

    }

    /\*\* Get the front element. \*/

    int peek() {

        if(s\_out.empty()) move();

        return s\_out.top();

    }

    /\*\* Returns whether the queue is empty. \*/

    bool empty() {

        return s\_in.empty() && s\_out.empty();

    }

};

/\*\*

 \* Your MyQueue object will be instantiated and called as such:

 \* MyQueue\* obj = new MyQueue();

 \* obj->push(x);

 \* int param\_2 = obj->pop();

 \* int param\_3 = obj->peek();

 \* bool param\_4 = obj->empty();

 \*/

## 225 用队列实现栈

class MyStack {

private:

    queue<int> q;

    int top\_elem;

    void move() {

        int size = q.size();

        while (size-- > 2) {

            q.push(q.front());

            q.pop();

        }

        top\_elem = q.front();

        q.push(q.front());

        q.pop();

    }

public:

    /\*\* Initialize your data structure here. \*/

    MyStack() {

        top\_elem = 0;

    }

    /\*\* Push element x onto stack. \*/

    void push(int x) {

        q.push(x);

        top\_elem = x;

    }

    /\*\* Removes the element on top of the stack and returns that element. \*/

    int pop() {

        int ret = top\_elem;

        move();

        q.pop();

        return ret;

    }

    /\*\* Get the top element. \*/

    int top() {

        return top\_elem;

    }

    /\*\* Returns whether the stack is empty. \*/

    bool empty() {

        return q.empty();

    }

};

/\*\*

 \* Your MyStack object will be instantiated and called as such:

 \* MyStack\* obj = new MyStack();

 \* obj->push(x);

 \* int param\_2 = obj->pop();

 \* int param\_3 = obj->top();

 \* bool param\_4 = obj->empty();

 \*/

# 滑动窗口专题

## 76 最小覆盖子串

class Solution {

public:

    string minWindow(string s, string t) {

        map<char, int> need, window;

        int valid = 0;

        for(auto c : t) ++need[c];

        int left = 0, right = 0;

        int start = 0, len = INT\_MAX;

        while(right < s.length()) {

            char rc = s[right];

            ++right;

            if(need.count(rc)) {

                ++window[rc];

                if(window[rc] == need[rc]) {

                    ++valid;

                }

            }

            while(valid == need.size()) {

                // 记录

                if(right-left < len) {

                    start = left;

                    len = right-left;

                }

                char lc = s[left];

                ++left;

                if(need.count(lc)) {

                    if(window[lc] == need[lc]) {

                        --valid;

                    }

                    --window[lc];

                }

            }

        }

        return len == INT\_MAX ? "" : s.substr(start, len);

    }

};

## 567 字符串的排列

class Solution {

public:

    bool checkInclusion(string s1, string s2) {

        map<char, int> need, window;

        int left = 0, right = 0;

        int valid = 0;

        for(auto c : s1) ++need[c];

        while(right < s2.size()) {

            char rc = s2[right];

            ++right;

            if(need.count(rc)) {

                ++window[rc];

                if(window[rc] == need[rc]) {

                    ++valid;

                }

            }

            while(right-left >= s1.size()) {

                // 全局判断

                if(valid == need.size())

                    return true;

                char lc = s2[left];

                ++left;

                // 更新窗口

                if(need.count(lc)) {

                    if(need[lc] == window[lc]) {

                        --valid;

                    }

                    --window[lc];

                }

            }

        }

        return false;

    }

};

## 438 找到字符串中所有字母异位词

class Solution {

public:

vector<int> findAnagrams(string s, string p) {

map<char, int> need, window;

int left = 0, right = 0;

int valid = 0;

vector<int> ans;

for(auto c : p) ++need[c];

while(right < s.size()) {

char rc = s[right];

++right;

if(need.count(rc)) {

++window[rc];

if(window[rc] == need[rc]) {

++valid;

}

}

while(right-left >= p.size()) {

// 更新

if(valid == need.size()) {

ans.push\_back(left);

}

char lc = s[left];

++left;

if(need.count(lc)) {

if(window[lc] == need[lc]) {

--valid;

}

--window[lc];

}

}

}

return ans;

}

};

## 3 无重复字符的最长子串

class Solution {public:

int lengthOfLongestSubstring(string s) {

int left = 0, right = 0;

map<char, int> window;

int res = 0;

while(right < s.size()) {

char rc = s[right];

++right;

++window[rc];

while(window[rc] > 1) {

char lc = s[left];

++left;

--window[lc];

}

res = max(res, right-left);

}

return res;

}

};

# 双指针

## 141 环形链表

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/class Solution {public:

bool hasCycle(ListNode \*head) {

ListNode\* fast = head, \*slow = head;

while(fast != nullptr && fast->next != nullptr) {

fast = fast->next->next;

slow = slow->next;

if(fast == slow) return true;

}

return false;

}

};

## 142 环形链表 II

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/class Solution {public:

ListNode \*detectCycle(ListNode \*head) {

bool has\_cycle = false;

ListNode \*fast = head, \*slow = head;

while(fast != nullptr && fast->next != nullptr) {

fast = fast->next->next;

slow = slow->next;

if(fast == slow) {

has\_cycle = true;

break;

}

}

if(!has\_cycle) return nullptr;

fast = head;

while(fast!=slow) {

fast = fast->next;

slow = slow->next;

}

return slow;

}

};

## 876 链表的中间结点

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/class Solution {public:

ListNode\* middleNode(ListNode\* head) {

ListNode\* fast = head, \*slow = head;

while(fast != nullptr && fast->next != nullptr) {

fast = fast->next->next;

slow = slow->next;

}

return slow;

}

};

## 剑指 Offer 22 链表中倒数第k个节点

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/class Solution {public:

ListNode\* getKthFromEnd(ListNode\* head, int k) {

ListNode \*fast = head, \*slow = head;

while(k-->0) {

fast = fast->next;

}

while(fast != nullptr) {

fast = fast->next;

slow = slow->next;

}

return slow;

}

};

## 167 两数之和 II - 输入有序数组

class Solution {public:

vector<int> twoSum(vector<int>& numbers, int target) {

int left = 1, right = numbers.size();

while(left < right) {

int lval = numbers[left-1], rval = numbers[right-1];

if(lval + rval == target) {

return vector<int>{left, right};

}

else if(lval + rval < target) {

++left;

}

else {

--right;

}

}

return vector<int>();

}

};

## 344 反转字符串

class Solution {public:

void reverseString(vector<char>& s) {

int left = 0, right = s.size()-1;

while(left < right) {

swap(s[left++], s[right--]);

}

}

};

# 二分查找

## 875 爱吃香蕉的珂珂

class Solution {

private:

    int getMax(vector<int>& piles) {

        int max\_elem = INT\_MIN;

        for(const auto n : piles) {

            max\_elem = max(max\_elem, n);

        }

        return max\_elem;

    }

    bool canFinish(vector<int>& piles, int H, int speed) {

        int time = 0;

        for(auto n : piles) {

            time += n/speed + ((n%speed==0) ? 0:1);

        }

        return time <= H;

    }

public:

    int minEatingSpeed(vector<int>& piles, int H) {

        int max\_elem = getMax(piles);

        int left = 1, right = max\_elem+1;

        while(left < right) {

            int mid = left + (right-left) / 2;

            if(canFinish(piles, H, mid)) {

                right = mid;

            }

            else {

                left = mid + 1;

            }

        }

        return left;

    }

};

## 1011 在 D 天内送达包裹的能力

class Solution {

private:

    int needDays(vector<int>& weights, int ability) {

        int days = 1;

        int sum = 0;

        for(int i=0; i<weights.size(); ++i) {

            if(sum + weights[i] <= ability) {

                sum += weights[i];

            }

            else {

                sum = weights[i];

                ++days;

            }

        }

        return days;

    }

    int getMax(vector<int>& weights) {

        int ret = INT\_MIN;

        for(auto w : weights) {

            ret = max(ret, w);

        }

        return ret;

    }

public:

    int shipWithinDays(vector<int>& weights, int D) {

        int sum = accumulate(weights.begin(), weights.end(), 0);

        int left = getMax(weights), right = sum+1;

        while(left < right) {

            int mid = left + (right-left)/2;

            if(needDays(weights, mid) <= D) {

                right = mid;

            }

            else {

                left = mid + 1;

            }

        }

        return left;

    }

};