

一、C++11 tips

```
// 1. 容器—vector
#include<vector>
vector<int> a(10) | vector<string> b | vector c(5, 1) // 初始化, a共10个内存空间, c是5个1
vector<int> (3, 1) | vector<int> {1, 2, 3} // 其他初始化方式
vector<int>(a.begin(), a.end()) // 用a赋值, a可以是vector<int>, 也可以是unordered_set<int>
.push_back(5) | .pop_back() // 尾部添加或删除元素
.size() | .resize(10) // 返回vector大小, 重新定义大小
.back() | .front() // 返回vector首部或尾部元素的引用
.begin() | .end() // 返回vector<int>::iterator, 用于遍历
.insert(a.begin(), 5) | .insert(a.begin() + 1, 3, 5) // 位置0插入5, 位置1插入3个5, 返回插入位置的迭代器
.erase(a.begin()) | .erase(a.begin() + 1, a.begin() + 3) // 删除a的0位置元素, 删除a的1~3位置元素, 返回删除位置的迭代器
```

```
// 2. 哈希集—unordered_set
#include<unordered_set>
unordered_set<int> hash_set // 初始化
.insert(5) | .erase(4) // 插入、删除
.count(5) | .size() // 对某元素计数、返回哈希集大小
.begin() | .end() // 返回iterator, 用于遍历
.clear() | .empty() // 清空、判断是否为空
```

```
// 3. 哈希映射—unordered_map
#include<unordered_map>
unordered_map<int, int> hash_map | hash_map[1] = 2 // 初始化、为空或不空均可直接赋值,
.insert(make_pair(1, 2)) | .erase(4) // 插入 (make_pair直接用) 删除 (4为key)
.count(5) | .size() // 对某元素计数 (5为key)、返回哈希集大小
.begin() | .end() // 返回iterator, 用于遍历
.clear() | .empty() // 清空、判断是否为空
for (unordered_map<int, int>::iterator it = hash_map.begin(); it != hash_map.end(); it++) {
    cout << *it.first << *it.second << it->first << it->second << endl;
}
```

```
// 3.1 映射—map, 用法同unordered_map
#include<map>
map<int, int> m | m[1] = 2
m[i].emplace_back(2)
```

```
// 4. 字符串—string
```

```

#include<string>
string s = "abc" | string s(2, 'c') | to_string(123) // 初始化
.size() | .length() // 返回长度，尽量用size, length返回

无符号

.resize() // 调整字符串长度
.substr(5) | .substr(5, 3) // 取子串，从5截到尾、从5往后截3个
.push_back('a') | .append('b') | += 'a' // 末尾追加
.insert(0, "abc") | .insert(1, "abc", 1, 2) // 0位置插入abc、1位置插入bc
.pop_back() | .erase(0, 1) // 末尾删除、删除位置0开始数1个字符
.replace(1, 2, "abc") // 位置1开始数2个字符替换为abc
.begin() | .end() // 返回iterator, 用于遍历
string s = "abc"; s[1] = 'm' // 可以直接赋值改变

// 字符串转数字
#include<cstdlib>
stoi("123") | stol("123") | stoll("123") // 转int、long、long long

// 字符串比大小
str1.compare(str2) | strcmp(str1, str2) // 逐位比大小，按照ascii码

// 大小写转换
#include<cctype>
tolower("ABC") | toupper("abc") // 大转小、小转大

```

```

// 5.栈和队列—stack、queue
#include<stack> | #include<queue>
.size() // 求大小均是size
.push() | .pop() | .empty() | .top() // 栈
.push() | .pop() | .empty() | .front() | .back() // 队列

// 优先队列
#include<priority_queue>
priority_queue<int> | priority_queue<int, less<int>()> // 队头大、队头小

```

```

// 6.各类算法函数—algorithm
#include<algorithm>
sort(起始地址, 结束地址, 排序规则) | less或great<类型>() // 不设置规则默认小到大，即less
sort(~, ~, cmp) | static bool cmp(a, b) {return a > b} // 自定义规则，此处是降序，必须加static
reserve(起始地址, 结束地址) // 转换顺序
copy(a起始地址, a结束地址, b起始地址) // a复制到b
find(起始地址, 结束地址, 10) // 找到10返回指针，找不到返回.end()
count(起始地址, 结束地址, 查找字符) // 返回字符串中查找字符的数量

```

```
// 7. 输入输出—printf、scanf
printf("%s%d", str, num) // 输出字符串和数字，不需要地址
scanf("%s%d", str, &num) // 输入字符串和数字，需要地址，str本身
地址
%s、%c、%d、%f、%lf // string、char、int、float、
double
%04d、%.4f // 不足4位补0、保留4位小数
```

```
// 8. 常用函数
```

```
// 8.1 accumulate
#include<numeric>
accumulate(起始地址, 结束地址, 累加初值)

// 8.2 memset
#include<string.h>
dp[m][n] —————> memset(dp, -1或0或false, m*n或sizeof(dp)) // 填1会出问题，一般填0、-1、
false

// 8.3 is系列
isalpha() | isupper() | isdigit() | islower() // 是否字母、大写、数字、小写，返回0或
1
```

二、数组

2.1 二分查找 (704、35、34、69、367)

- 704

704. 二分查找

难度 简单

1304

收藏

分享

切换为英文

接收动态

反馈

给定一个 n 个元素有序的（升序）整型数组 `nums` 和一个目标值 `target`，写一个函数搜索 `nums` 中的 `target`，如果目标值存在返回下标，否则返回 `-1`。

示例 1：

输入： `nums = [-1,0,3,5,9,12]`, `target = 9`

输出： 4

解释： 9 出现在 `nums` 中并且下标为 4

示例 2：

输入： `nums = [-1,0,3,5,9,12]`, `target = 2`

输出： -1

解释： 2 不存在 `nums` 中因此返回 -1

提示：

1. 你可以假设 `nums` 中的所有元素是不重复的。
2. n 将在 $[1, 10000]$ 之间。
3. `nums` 的每个元素都将在 $[-9999, 9999]$ 之间。

The screenshot shows a code editor interface with a dark theme. At the top, there is a toolbar with icons for file operations and a mode switch labeled "智能模式". Below the toolbar, the code editor displays a C++ class definition:

```
1 class Solution {
2 public:
3     int search(vector<int>& nums, int target) {
4         int left = 0, right = nums.size() - 1;
5         while (left <= right) {
6             int mid = left + (right - left) / 2;
7             if (nums[mid] == target) return mid;
8             else if (nums[mid] > target) right = mid - 1;
9             else left = mid + 1;
10        }
11        return -1;
12    }
13};
```

35. 搜索插入位置

难度 简单 2010 年 收藏 分享 切换为英文 接收动态 反馈

给定一个排序数组和一个目标值，在数组中找到目标值，并返回其索引。如果目标值不存在于数组中，返回它将被按顺序插入的位置。

请必须使用时间复杂度为 $O(\log n)$ 的算法。

示例 1:

输入: `nums = [1,3,5,6], target = 5`
输出: 2

示例 2:

输入: `nums = [1,3,5,6], target = 2`
输出: 1

提示:

- $1 \leq \text{nums.length} \leq 10^4$
- $-10^4 \leq \text{nums}[i] \leq 10^4$
- `nums` 为无重复元素的升序排列数组
- $-10^4 \leq \text{target} \leq 10^4$

```
① C++ ② 智能模式 ③ 模拟面试 | i P ⌂ >_ ⚙ [ ]  
1 class Solution {  
2 public:  
3     int searchInsert(vector<int>& nums, int target) {  
4         int left = 0, right = nums.size() - 1;  
5         while (left <= right) {  
6             int mid = left + ((right - left) >> 1);  
7             if (nums[mid] == target) {  
8                 // 题目是无重复，此处消除重复影响  
9                 // while (mid > 0 && nums[mid] == nums[mid - 1]) mid -= 1;  
10                return mid;  
11            }  
12            else if (nums[mid] > target) right = mid - 1;  
13            else left = mid + 1;  
14        }  
15    }  
16    return right + 1;  
17 }
```

34. 在排序数组中查找元素的第一个和最后一个位置

难度 中等 2287 收藏 分享 切换为英文 接收动态 反馈

给你一个按照非递减顺序排列的整数数组 `nums`，和一个目标值 `target`。请你找出给定目标值在数组中的开始位置和结束位置。

如果数组中不存在目标值 `target`，返回 `[-1, -1]`。

你必须设计并实现时间复杂度为 $O(\log n)$ 的算法解决此问题。

示例 1：

输入: `nums = [5,7,7,8,8,10]`, `target = 8`

输出: `[3,4]`

示例 2：

输入: `nums = [5,7,7,8,8,10]`, `target = 6`

输出: `[-1,-1]`

示例 3：

输入: `nums = []`, `target = 0`

输出: `[-1,-1]`

提示：

- $0 \leq \text{nums.length} \leq 10^5$
- $-10^9 \leq \text{nums}[i] \leq 10^9$
- `nums` 是一个非递减数组
- $-10^9 \leq \text{target} \leq 10^9$

```
① C++ ② 智能模式 ③ 模拟面试 | i P ⌂ >_ ⚙ [ ]  
1 class Solution {  
2 public:  
3     vector<int> searchRange(vector<int>& nums, int target) {  
4         int left = 0, right = nums.size() - 1;  
5         while (left <= right) {  
6             int mid = left + ((right - left) >> 1);  
7             if (nums[mid] == target) {  
8                 int res_l = mid, res_r = mid;  
9                 while (res_l > 0 && nums[res_l] == nums[res_l - 1]) res_l -= 1;  
10                while (res_r < nums.size() - 1 && nums[res_r] == nums[res_r + 1])  
11                    res_r += 1;  
12                return vector<int> {res_l, res_r};  
13            } else if (nums[mid] > target) right = mid - 1;  
14            else left = mid + 1;  
15        }  
16    }  
17 }  
18 };
```

- 69

69. x 的平方根

难度 简单 1331 收藏 分享 切换为英文 接收动态 反馈

给你一个非负整数 x ，计算并返回 x 的 算术平方根。

由于返回类型是整数，结果只保留 整数部分，小数部分将被 舍去。

注意：不允许使用任何内置指数函数和算符，例如 `pow(x, 0.5)` 或者 `x ** 0.5`。

示例 1：

输入: $x = 4$

输出: 2

示例 2：

输入: $x = 8$

输出: 2

解释: 8 的算术平方根是 $2.82842\dots$ ，由于返回类型是整数，小数部分将被舍去。

提示：

- $0 \leq x \leq 2^{31} - 1$

```
i C++ • 智能模式  
1 class Solution {  
2 public:  
3     int mySqrt(int x) {  
4         if (x == 0) return 0;  
5         int temp = x;  
6         while (x / temp < temp) {  
7             temp = temp / 2;  
8         }  
9         if (x / temp == temp) return temp;  
10        while (x / temp >= temp) temp++;  
11        return temp - 1;  
12    }  
13 }
```

- 367

367. 有效的完全平方数

难度 简单 494 收藏 分享 切换为英文 接收动态 反馈

给你一个正整数 `num`。如果 `num` 是一个完全平方数，则返回 `true`，否则返回 `false`。

完全平方数 是一个可以写成某个整数的平方的整数。换句话说，它可以写成某个整数和自身的乘积。

不能使用任何内置的库函数，如 `sqrt`。

示例 1:

输入: `num = 16`

输出: `true`

解释: 返回 `true`，因为 $4 * 4 = 16$ 且 4 是一个整数。

示例 2:

输入: `num = 14`

输出: `false`

解释: 返回 `false`，因为 $3.742 * 3.742 = 14$ 但 3.742 不是一个整数。

提示:

- $1 \leq num \leq 2^{31} - 1$

① C++ • 智能模式

模拟面试 | i P ⌂ ⌄ ⌅ ⌆ ⌇

```
1 class Solution {
2 public:
3     bool isPerfectSquare(int num) {
4         if (num == 1) return true;
5         int left = 0, right = num;
6         while (left <= right) {
7             int mid = left + ((right - left) >> 1);
8             if (num / mid < mid) right = mid - 1;
9             else if (mid * mid == num) return true;
10            else left = mid + 1;
11        }
12        return false;
13    }
14};
```

2.2 移除元素（27、26、283、844）

- 27

27. 移除元素

难度 简单 1792 收藏 分享 切换为英文 接收动态 反馈

给你一个数组 `nums` 和一个值 `val`，你需要 **原地** 移除所有数值等于 `val` 的元素，并返回移除后数组的新长度。

不要使用额外的数组空间，你必须仅使用 $O(1)$ 额外空间并 **原地** **修改输入数组**。

元素的顺序可以改变。你不需要考虑数组中超出新长度后面的元素。

示例 1：

输入: `nums = [3,2,2,3]`, `val = 3`

输出: 2, `nums = [2,2]`

解释: 函数应该返回新的长度 2，并且 `nums` 中的前两个元素均为 2。你不需要考虑数组中超出新长度后面的元素。例如，函数返回的新长度为 2，而 `nums = [2,2,3,3]` 或 `nums = [2,2,0,0]`，也会被视作正确答案。

示例 2：

输入: `nums = [0,1,2,2,3,0,4,2]`, `val = 2`

输出: 5, `nums = [0,1,4,0,3]`

解释: 函数应该返回新的长度 5，并且 `nums` 中的前五个元素为 0, 1, 3, 0, 4。注意这五个元素可为任意顺序。你不需要考虑数组中超出新长度后面的元素。

提示:

- $0 \leq \text{nums.length} \leq 100$
- $0 \leq \text{nums[i]} \leq 50$
- $0 \leq \text{val} \leq 100$

The screenshot shows a code editor interface with a dark theme. At the top, there are tabs for 'C++' and '智能模式'. On the right side, there are various icons for simulation, interview mode, and other tools. The main area contains the following C++ code:

```
1 class Solution {
2 public:
3     int removeElement(vector<int>& nums, int val) {
4         int temp = 0;
5         for (int i = 0; i < nums.size(); i++) {
6             if (nums[i] != val) nums[temp++] = nums[i];
7         }
8         // for (int j = temp; j < nums.size(); j++) {
9         //     nums[j] = 0;
10        //}
11        return temp;
12    }
13};
```

26. 删除有序数组中的重复项

难度 简单 3158 收藏 分享 切换为英文 接收动态 反馈

给你一个升序排列的数组 `nums`，请你原地删除重复出现的元素，使每个元素只出现一次，返回删除后数组的新长度。元素的相对顺序应该保持一致。然后返回 `nums` 中唯一元素的个数。

考虑 `nums` 的唯一元素的数量为 `k`，你需要做以下事情确保你的题解可以被通过：

- 更改数组 `nums`，使 `nums` 的前 `k` 个元素包含唯一元素，并按照它们最初在 `nums` 中出现的顺序排列。`nums` 的其余元素与 `nums` 的大小不重要。
- 返回 `k`。

示例 1：

输入: `nums = [1,1,2]`

输出: `2, nums = [1,2,_]`

解释: 函数应该返回新的长度 `2`，并且原数组 `nums` 的前两个元素被修改为 `1, 2`。不需要考虑数组中超出新长度后面的元素。

示例 2：

输入: `nums = [0,0,1,1,1,2,2,3,3,4]`

输出: `5, nums = [0,1,2,3,4]`

解释: 函数应该返回新的长度 `5`，并且原数组 `nums` 的前五个元素被修改为 `0, 1, 2, 3, 4`。不需要考虑数组中超出新长度后面的元素。

提示:

- `1 <= nums.length <= 3 * 104`
- `-104 <= nums[i] <= 104`
- `nums` 已按升序排列

```
① C++ 模拟面试 i P ⌂ > ⌂ ⌂
1 class Solution {
2 public:
3     int removeDuplicates(vector<int>& nums) {
4         if (nums.size() <= 1) return nums.size();
5         int temp = 0;
6         for (int i = 0; i < (nums.size() - 1); i++) {
7             if (nums[i] != nums[i+1]) nums[temp++] = nums[i];
8         }
9         nums[temp] = nums[nums.size()-1];
10        return temp + 1;
11    }
12};
```

283. 移动零

难度 简单  收藏  分享  切换为英文  接收动态  反馈

给定一个数组 `nums`，编写一个函数将所有 `0` 移动到数组的末尾，同时保持非零元素的相对顺序。

请注意，必须在不复制数组的情况下原地对数组进行操作。

示例 1：

输入: `nums = [0,1,0,3,12]`

输出: `[1,3,12,0,0]`

示例 2：

输入: `nums = [0]`

输出: `[0]`

提示:

- $1 \leq \text{nums.length} \leq 10^4$
- $-2^{31} \leq \text{nums}[i] \leq 2^{31} - 1$



The code editor interface shows a C++ file named `Solution.cpp`. The code implements a function `moveZeroes` that takes a reference to a vector of integers `nums`. It first counts the number of zeros in the array. Then it iterates through the array, placing non-zero values at the front and zeros at the end. Finally, it sets the last `count` elements to zero.

```
1 class Solution {
2 public:
3     void moveZeroes(vector<int>& nums) {
4         int count = 0;
5         for (int i = 0; i < nums.size(); i++) {
6             if (nums[i] == 0) count++;
7             else nums[i - count] = nums[i];
8         }
9         for (int j = nums.size() - count; j < nums.size(); j++) {
10            nums[j] = 0;
11        }
12    }
13 }
```

844. 比较含退格的字符串

难度 简单 收藏 分享 切换为英文 接收动态 反馈

给定 `s` 和 `t` 两个字符串，当它们分别被输入到空白的文本编辑器后，如果两者相等，返回 `true`。`#` 代表退格字符。

注意：如果对空文本输入退格字符，文本继续为空。

示例 1：

输入：`s = "ab#c"`, `t = "ad#c"`

输出：`true`

解释：`s` 和 `t` 都会变成 `"ac"`。

示例 2：

输入：`s = "ab##"`, `t = "c#d#"`

输出：`true`

解释：`s` 和 `t` 都会变成 `""`。

示例 3：

输入：`s = "a#c"`, `t = "b"`

输出：`false`

解释：`s` 会变成 `"c"`, 但 `t` 仍然是 `"b"`。

提示：

- `1 <= s.length, t.length <= 200`
- `s` 和 `t` 只含有小写字母以及字符 `'#'`

• 智能模式

• 模拟面试 | i P ↻ ⌂ ⌂ ⌂

```
1 #include<stack>
2 class Solution {
3 public:
4     bool backspaceCompare(string s, string t) {
5         stack<char> news, newt;
6         for (int i = 0; i < s.length(); i++) {
7             if (s[i] != '#') news.push(s[i]);
8             else if (s[i] == '#' && !news.empty()) news.pop();
9         }
10        string s_new = "";
11        while (!news.empty()) {
12            s_new = news.top() + s_new;
13            news.pop();
14        }
15        for (int i = 0; i < t.length(); i++) {
16            if (t[i] != '#') newt.push(t[i]);
17            else if (t[i] == '#' && !newt.empty()) newt.pop();
18        }
19        string t_new = "";
20        while (!newt.empty()) {
21            t_new = newt.top() + t_new;
22            newt.pop();
23        }
24        return s_new == t_new;
25    }
26};
```

2.3 有序数组的平方 (977)

• 977

977. 有序数组的平方

难度 简单 798 收藏 分享 切换为英文 接收动态 反馈

给你一个按 非递减顺序 排序的整数数组 `nums`，返回 每个数字的平方 组成的新数组，要求也按 非递减顺序 排序。

示例 1：

输入: `nums = [-4, -1, 0, 3, 10]`

输出: `[0, 1, 9, 16, 100]`

解释: 平方后, 数组变为 `[16, 1, 0, 9, 100]`

排序后, 数组变为 `[0, 1, 9, 16, 100]`

示例 2：

输入: `nums = [-7, -3, 2, 3, 11]`

输出: `[4, 9, 9, 49, 121]`

提示:

- $1 \leq \text{nums.length} \leq 10^4$
- $-10^4 \leq \text{nums}[i] \leq 10^4$
- `nums` 已按 非递减顺序 排序

```
1 class Solution {
2 public:
3     vector<int> sortedSquares(vector<int>& nums) {
4         int p = -1, q = -1;
5         for (int i = 0; i < nums.size(); i++) {
6             if (nums[i] >= 0) {
7                 p = i - 1;
8                 q = i;          找到第一个非负的数字
9                 break;
10            }
11        }
12        if (q == 0) {
13            vector<int> result;
14            for (int i = 0; i < nums.size(); i++) {    全非负情况
15                result.push_back(nums[i]*nums[i]);
16            }
17            return result;
18        }
19        else if (q == -1 || nums[nums.size()-1] == 0) {
20            vector<int> result;
21            for (int i = nums.size() - 1; i >= 0; i--) {   全负情况 / 全非正情况
22                result.push_back(nums[i]*nums[i]);
23            }
24            return result;
25        }
26        vector<int> result;
27        while(p >= 0 && q < nums.size()) {
28            if (nums[p]*nums[p] > nums[q]*nums[q]) {
29                result.push_back(nums[q]*nums[q]);      从中间向两边扩散，小的放入result
30                q++;
31            } else {
32                result.push_back(nums[p]*nums[p]);
33                p--;
34            }
35            cout << 'p' << p << endl;
36            cout << 'q' << q << endl;
37        }
38        while(p >= 0) {
39            result.push_back(nums[p]*nums[p]);
40            p--;
41        }
42        while(q < nums.size()) {
43            result.push_back(nums[q]*nums[q]);
44            q++;
45        }
46        return result;
47    }
48};
```

```
1 class Solution {  
2     public:  
3         vector<int> sortedSquares(vector<int>& A) {  
4             int k = A.size() - 1;  
5             vector<int> result(A.size(), 0);  
6             for (int i = 0, j = A.size() - 1; i <= j;) { // 注意这里要i <= j, 因为最后要从两侧往中间填入  
7                 if (A[i] * A[i] < A[j] * A[j]) {  
8                     result[k--] = A[j] * A[j];  
9                     j--;  
10                }  
11                else {  
12                    result[k--] = A[i] * A[i];  
13                    i++;  
14                }  
15            }  
16            return result;  
17        }  
18    };
```

2.4 长度最小的子数组 (209、904、76)

- 209

209. 长度最小的子数组

难度 中等

1701

收藏

分享

切换为英文

接收动态

反馈

给定一个含有 n 个正整数的数组和一个正整数 $target$ 。

找出该数组中满足其和 $\geq target$ 的长度最小的 连续子数组 $[nums_1, nums_{1+1}, \dots, nums_{r-1}, nums_r]$ ，并返回其长度。如果不存在符合条件的子数组，返回 0。

示例 1：

输入: target = 7, nums = [2,3,1,2,4,3]

输出: 2

解释: 子数组 [4,3] 是该条件下的长度最小的子数组。

示例 2：

输入: target = 4, nums = [1,4,4]

输出: 1

示例 3：

输入: target = 11, nums = [1,1,1,1,1,1,1,1]

输出: 0

提示：

- $1 \leq target \leq 10^9$
- $1 \leq nums.length \leq 10^5$
- $1 \leq nums[i] \leq 10^5$

```
1 class Solution {
2 public:
3     int minSubArrayLen(int target, vector<int>& nums) {
4         int result = nums.size() + 1;
5         int i = 0, j = 0, acc = nums[0];
6         while (j < nums.size()) {
7             if (acc >= target) {
8                 result = (j - i + 1) >= result ? result : (j - i + 1);
9                 acc -= nums[i++];
10            } else {
11                j++;
12                if (j == nums.size()) break;
13                acc += nums[j];
14            }
15        }
16        return (result == (nums.size() + 1)) ? 0 : result;
17    }
18 }
```

滑动窗口 [i,j]

1. 窗内和大于等于target, 缩i
2. 窗内和小于target, 扩j

• 904

904. 水果成篮

难度 中等  487  收藏  分享  切换为英文  接收动态  反馈

你正在探访一家农场，农场从左到右种植了一排果树。这些树用一个整数数组 `fruits` 表示，其中 `fruits[i]` 是第 `i` 棵树上的水果 **种类**。

你想要尽可能多地收集水果。然而，农场的主人设定了一些严格的规矩，你必须按照要求采摘水果：

- 你只有 **两个** 篮子，并且每个篮子只能装 **单一类型** 的水果。每个篮子能够装的水果总量没有限制。
- 你可以选择任意一棵树开始采摘，你必须从 **每棵** 树（包括开始采摘的树）上 **恰好摘一个水果**。采摘的水果应当符合篮子中的水果类型。每采摘一次，你将会向右移动到下一棵树，并继续采摘。
- 一旦你走到某棵树前，但水果不符合篮子的水果类型，那么就必须停止采摘。

给你一个整数数组 `fruits`，返回你可以收集的水果的 **最大** 数目。

示例 1：

输入: `fruits = [1,2,1]`

输出: 3

解释: 可以采摘全部 3 棵树。

示例 2：

输入: `fruits = [0,1,2,2]`

输出: 3

解释: 可以采摘 [1,2,2] 这三棵树。

如果从第一棵树开始采摘，则只能采摘 [0,1] 这两棵树。

示例 3：

输入: `fruits = [1,2,3,2,2]`

输出: 4

解释: 可以采摘 [2,3,2,2] 这四棵树。

如果从第一棵树开始采摘，则只能采摘 [1,2] 这两棵树。

示例 4：

输入: `fruits = [3,3,3,1,2,1,1,2,3,3,4]`

输出: 5

解释: 可以采摘 [1,2,1,1,2] 这五棵树。

提示：

- `1 <= fruits.length <= 105`
- `0 <= fruits[i] < fruits.length`

```
1 #include<unordered_map>
2 class Solution {
3 public:
4     int totalFruit(vector<int>& fruits) {  
5         if (fruits.size() <= 2) return fruits.size();  
6         int result = 0;  
7         int i = 0, j = 0;  
8         unordered_map<int, int> m;  
9         m[fruits[0]] += 1;  
10        while (j < fruits.size()) {  
11            if (m.size() <= 2) {  
12                result = (result >= (j - i + 1)) ? result : (j - i + 1);  
13                j++;  
14                if (j == fruits.size()) continue;  
15                m[fruits[j]] += 1;  
16            } else {  
17                m[fruits[i]] -= 1;  
18                if (m[fruits[i]] == 0) m.erase(fruits[i]);  
19                i++;  
20            }  
21        }  
22        return result;  
23    }  
24};
```

滑动窗口 [i, j]

1. map的size小于等于2, 当前窗口满足条件, 更新result、map、j

2. map的size大于2, 当前窗口不满足条件, 更新map、i

76. 最小覆盖子串

难度 困难

2493

收藏

分享

切换为英文

接收动态

反馈

给你一个字符串 s 、一个字符串 t 。返回 s 中涵盖 t 所有字符的最小子串。如果 s 中不存在涵盖 t 所有字符的子串，则返回空字符串 `""`。

注意：

- 对于 t 中重复字符，我们寻找的子字符串中该字符数量必须不少于 t 中该字符数量。
- 如果 s 中存在这样的子串，我们保证它是唯一的答案。

示例 1：

输入: $s = "ADOBECODEBANC"$, $t = "ABC"$

输出: "BANC"

解释: 最小覆盖子串 "BANC" 包含来自字符串 t 的 'A'、'B' 和 'C'。

示例 2：

输入: $s = "a"$, $t = "a"$

输出: "a"

解释: 整个字符串 s 是最小覆盖子串。

示例 3：

输入: $s = "a"$, $t = "aa"$

输出: `""`

解释: t 中两个字符 'a' 均应包含在 s 的子串中，因此没有符合条件的子字符串，返回空字符串。

提示：

- $m == s.length$
- $n == t.length$
- $1 \leq m, n \leq 10^5$
- s 和 t 由英文字母组成

```
#include<unordered_map>
class Solution {
public:
    string minWindow(string s, string t) {
        if (s.size() < t.size()) return "";
        // tm维护s子串中还需要多少个某个字母才能满足条件
```

```

unordered_map<char, int> tm;
for (int i = 0; i < t.size(); i++) {
    tm[t[i]] += 1;
}
// if_cover维护当前子串是否满足条件，解决了每次都需要遍历一遍tm的问题
int left = 0, right = 0, if_cover = 0, min_length = s.size() + 1;
vector<int> result = vector<int> (2, -1);
for (; right < s.size(); right++) {
    // t中不含的字母，跳过
    if (tm.find(s[right]) == tm.end()) {
        continue;
    } else {
        // s子串中的s[right]数量等于t中s[right]数量, =0才++, s[right]第一次满足才++
        if (--tm[s[right]] == 0) if_cover++;
        // s子串已满足题目条件
        if (if_cover == tm.size()) {
            // 子串第一次满足条件时，更新min_length和result
            if (min_length > (right - left + 1)) {
                min_length = (right - left + 1);
                result[0] = left, result[1] = right;
            }
            // 左移left直至s子串不满足题目条件
            while (if_cover == tm.size()) {
                // 移除的字符在t中
                if (tm.find(s[left]) != tm.end()) {
                    // 移除后s子串中的s[left]已不满足题目条件，跳出while循环
                    if (++tm[s[left]] == 1) {
                        if_cover--;
                        left++;
                        continue;
                    }
                    // 移除后s子串中的s[left]仍满足题目条件，left先加，然后更新
                    else {
                        left++;
                        // 更新min_length和result
                        if (min_length > (right - left + 1)) {
                            min_length = (right - left + 1);
                            result[0] = left, result[1] = right;
                        }
                    }
                }
                // 移除的字符不在t中，left先加，然后更新
            } else {
                left++;
                // 更新min_length和result
                if (min_length > (right - left + 1)) {
                    min_length = (right - left + 1);
                    result[0] = left, result[1] = right;
                }
            }
        }
    }
}

```

```
        }
    }
}

if (min_length == (s.size() + 1)) return "";
return s.substr(result[0], result[1] - result[0] + 1);
}
};
```

2.5 螺旋矩阵 2 (59、54)

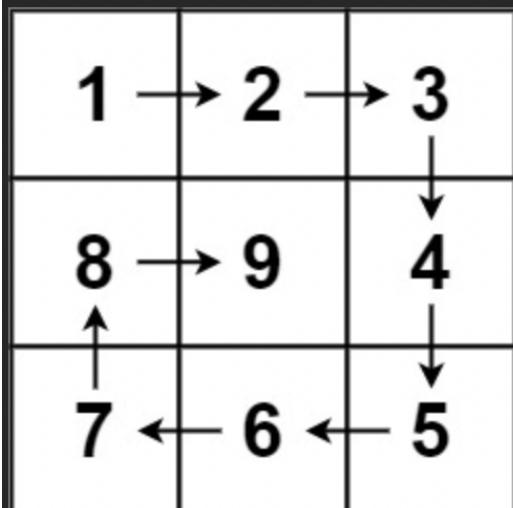
- 59

59. 螺旋矩阵 II

难度 中等 1050 收藏 分享 切换为英文 接收动态 反馈

给你一个正整数 n ，生成一个包含 1 到 n^2 所有元素，且元素按顺时针顺序螺旋排列的 $n \times n$ 正方形矩阵 `matrix`。

示例 1：



输入: $n = 3$

输出: `[[1,2,3],[8,9,4],[7,6,5]]`

示例 2：

输入: $n = 1$

输出: `[[1]]`

提示：

- $1 \leq n \leq 20$

```
① C++ • 智能模式 ② 模拟面试 | i P o >_ ⚙ [ ]  
1 class Solution {  
2 public:  
3     vector<vector<int>> generateMatrix(int n) {  
4         vector<vector<int>> result(n, vector<int>(n, 0));  
5         int l = 0, r = n - 1, t = 0, b = n - 1;  
6         int curNum = 1;  
7         while (curNum <= n*n) {  
8             for (int i = l; i <= r; i++) result[t][i] = curNum++;  
9             t++;  
10            for (int i = t; i <= b; i++) result[i][r] = curNum++;  
11            r--;  
12            for (int i = r; i >= l; i--) result[b][i] = curNum++;  
13            b--;  
14            for (int i = b; i >= t; i--) result[i][l] = curNum++;  
15            l++;  
16        }  
17        return result;  
18    }  
19};
```

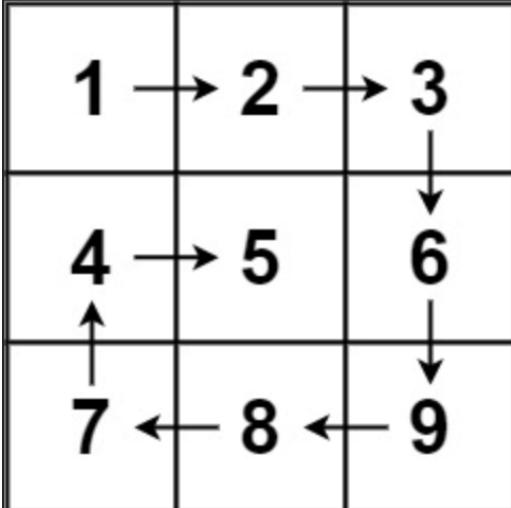
• 54

54. 螺旋矩阵

难度 中等 1376 收藏 分享 切换为英文 接收动态 反馈

给你一个 m 行 n 列的矩阵 matrix ，请按照 顺时针螺旋顺序，返回矩阵中的所有元素。

示例 1：

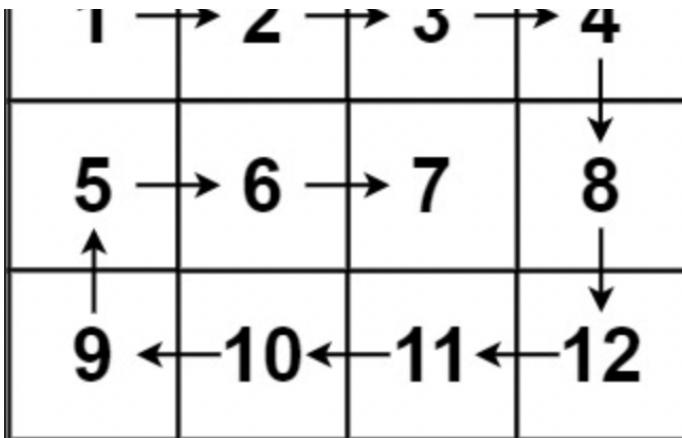


输入: $\text{matrix} = [[1,2,3],[4,5,6],[7,8,9]]$

输出: $[1,2,3,6,9,8,7,4,5]$

示例 2：





输入: matrix = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]

输出: [1,2,3,4,8,12,11,10,9,5,6,7]

提示:

- $m == \text{matrix.length}$
- $n == \text{matrix[i].length}$
- $1 \leq m, n \leq 10$
- $-100 \leq \text{matrix}[i][j] \leq 100$

```

i C++ 智能模式 模拟面试 i P ⌂ > ⌂ []
1 class Solution {
2 public:
3     vector<int> spiralOrder(vector<vector<int>>& matrix) {
4         vector<int> result;
5         int l = 0, r = matrix[0].size()-1, t = 0, b = matrix.size()-1;
6         while (l <= r && t <= b) {
7             for (int i = l; i <= r; i++) result.push_back(matrix[t][i]);
8             t++;
9             for (int i = t; i <= b; i++) result.push_back(matrix[i][r]);
10            r--;
11            for (int i = r; i >= l; i--) result.push_back(matrix[b][i]);
12            b--;
13            for (int i = b; i >= t; i--) result.push_back(matrix[i][l]);
14            l++;
15        }
16        int new_size = (matrix.size())*(matrix[0].size());
17        result.resize(new_size);
18        return result;
19    }
20}

```

解决不是正方形带来的答案不正确问题

三、链表

3.1 移除链表元素（203）

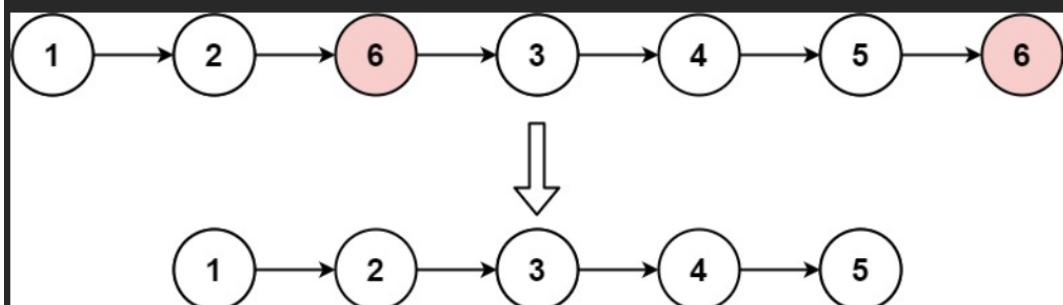
- 203

203. 移除链表元素

难度 简单 1247 收藏 分享 切换为英文 接收动态 反馈

给你一个链表的头节点 `head` 和一个整数 `val`，请你删除链表中所有满足 `Node.val == val` 的节点，并返回新的头节点。

示例 1：



输入: `head = [1,2,6,3,4,5,6], val = 6`

输出: `[1,2,3,4,5]`

示例 2：

输入: `head = [], val = 1`

输出: `[]`

示例 3：

输入: `head = [7,7,7,7], val = 7`

输出: `[]`

提示：

- 列表中的节点数目在范围 `[0, 104]` 内
- `1 <= Node.val <= 50`
- `0 <= val <= 50`

```
① C++ ② 智能模式 ③ 模拟面试 | i P ⌂ >_ ⚙ [ ]
```

```
1 // ...
2 * Definition for singly-linked list.
3 * struct ListNode {
4 *     int val;
5 *     ListNode *next;
6 *     ListNode() : val(0), next(nullptr) {}
7 *     ListNode(int x) : val(x), next(nullptr) {}
8 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
9 * };
10 */
11 class Solution {
12 public:
13     ListNode* removeElements(ListNode* head, int val) {
14         if (head == nullptr) return nullptr;
15         ListNode* p = head;
16         ListNode* q = head;
17         while (q) {
18             if (q->val == val) {
19                 q = q->next;
20                 p->next = q;
21             } else {
22                 p = q;
23                 q = q->next;
24             }
25         }
26         if(head->val == val) head = head->next;
27         return head;
28     }
29 };
```

解决第一个节点val就需要删除的情况

3.2 反转链表（206）

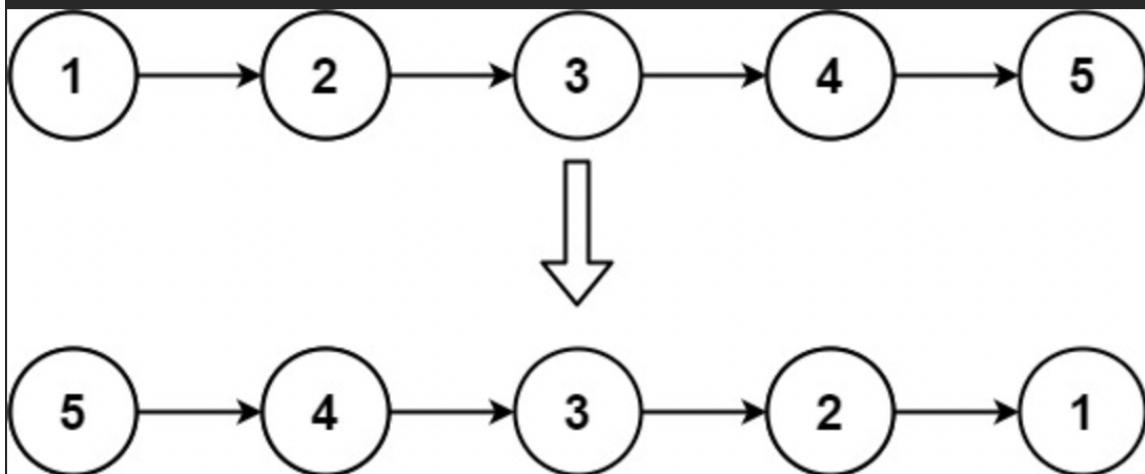
- 206

206. 反转链表

难度 简单 3146 收藏 分享 切换为英文 接收动态 反馈

给你单链表的头节点 `head`，请你反转链表，并返回反转后的链表。

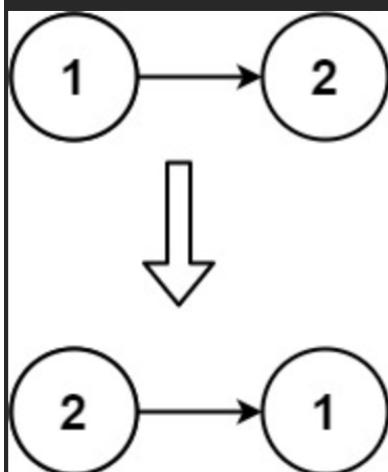
示例 1：



输入: `head = [1,2,3,4,5]`

输出: `[5,4,3,2,1]`

示例 2:



输入: head = [1,2]

输出: [2,1]

示例 3:

输入: head = []

输出: []

提示:

- 链表中节点的数目范围是 [0, 5000]
- $-5000 \leq \text{Node.val} \leq 5000$

```
1  /**
2  * Definition for singly-linked list.
3  * struct ListNode {
4  *     int val;
5  *     ListNode *next;
6  *     ListNode() : val(0), next(nullptr) {}
7  *     ListNode(int x) : val(x), next(nullptr) {}
8  *     ListNode(int x, ListNode *next) : val(x), next(next) {}
9  * };
10 */
11 class Solution {
12 public:
13     ListNode* reverseList(ListNode* head) {
14         if (head == nullptr || head->next == nullptr) return head;
15         ListNode* one = head;
16         ListNode* two = head->next;
17         ListNode* three = head->next->next;
18         one->next = nullptr;
19         while (two) {
20             three = two->next;
21             two->next = one;
22             one = two;
23             two = three;
24         }
25         head = one;
26         return head;
27     }
28 }
```

双指针法，当前two，前一个
one，three暂时保存下一个

```
1 class Solution {
2 public:
3     ListNode* reverse(ListNode* pre,ListNode* cur){
4         if(cur == NULL) return pre;
5         ListNode* temp = cur->next;
6         cur->next = pre;
7         // 可以和双指针法的代码进行对比，如下递归的写法，其实就是做了这两步
8         // pre = cur;
9         // cur = temp;
10        return reverse(cur,temp);
11    }
12    ListNode* reverseList(ListNode* head) {
13        // 和双指针法初始化是一样的逻辑
14        // ListNode* cur = head;
15        // ListNode* pre = NULL;
16        return reverse(NULL, head);
17    }
18 }
19 };
```

递归法

3.3 两两交换链表中的节点（24）

- 24

24. 两两交换链表中的节点

难度 中等

1822

收藏

分享

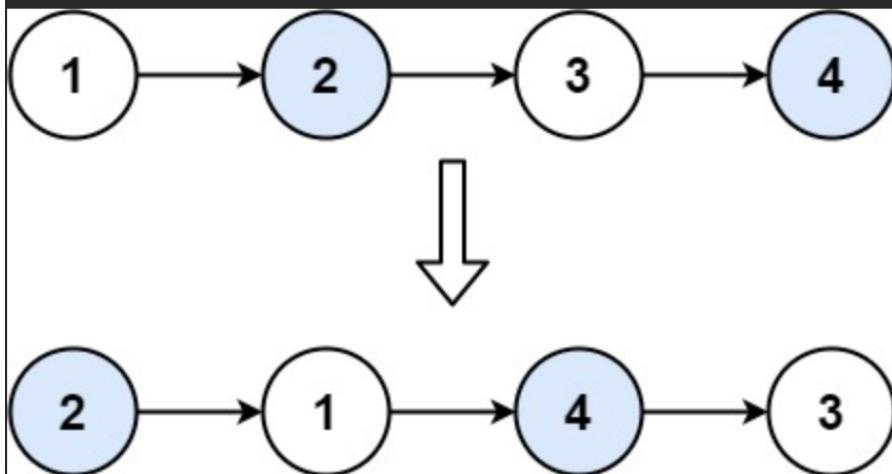
切换为英文

接收动态

反馈

给你一个链表，两两交换其中相邻的节点，并返回交换后链表的头节点。你必须在不修改节点内部的值的情况下完成本题（即，只能进行节点交换）。

示例 1：



输入: head = [1,2,3,4]

输出: [2,1,4,3]

示例 2：

输入: head = []

输出: []

示例 3：

输入: head = [1]

输出: [1]

提示：

- 链表中节点的数目在范围 `[0, 100]` 内

- $0 \leq \text{Node.val} \leq 100$

C++ 智能模式 模拟面试

```

9  */
10 */
11 class Solution {
12 public:
13     ListNode* swapPairs(ListNode* head) {
14         if (head == nullptr || head->next == nullptr) return head;
15         ListNode* new_head = head->next;      new_head保存新头节点
16         ListNode* one = head;
17         ListNode* two = head->next;
18         ListNode* three = head;
19         ListNode* temp;
20         while (two != nullptr && three != nullptr) {
21             temp = two->next;
22             one->next = two;
23             two->next = three;
24             one = three;
25             three = temp;
26             if (three) two = three->next;
27             else two = nullptr;
28         }
29         if (three) {
30             one->next = three;
31         } else {
32             one->next = nullptr;
33         }
34         head = new_head;
35         return head;
36     }
37 };

```

- temp保存后一对节点的前者
- one是已处理完的一对节点的前者
需要链接到后一对节点的后者
- two是后一对节点的后者
- three是后一对节点的前者

四、哈希表

4.1 哈希表理论基础

集合	底层实现	是否有序	数值是否可以重复	能否更改数值	查询效率	增删效率
std::set	红黑树	有序	否	否	O(log n)	O(log n)
std::multiset	红黑树	有序	是	否	O(logn)	O(logn)
std::unordered_set	哈希表	无序	否	否	O(1)	O(1)

映射	底层实现	是否有序	数值是否可以重复	能否更改数值	查询效率	增删效率
std::map	红黑树	key有序	key不可重复	key不可修改	O(logn)	O(logn)
std::multimap	红黑树	key有序	key可重复	key不可修改	O(log n)	O(log n)
std::unordered_map	哈希表	key无序	key不可重复	key不可修改	O(1)	O(1)

4.2 有效的字母异位词 (242、383、49、438)

- 242

242. 有效的字母异位词

难度 简单 783 收藏 复制 文章 分享

给定两个字符串 s 和 t ，编写一个函数来判断 t 是否是 s 的字母异位词。

注意：若 s 和 t 中每个字符出现的次数都相同，则称 s 和 t 互为字母异位词。

示例 1：

输入: $s = \text{"anagram"}, t = \text{"nagaram"}$

输出: true

示例 2：

输入: $s = \text{"rat"}, t = \text{"car"}$

输出: false

提示:

- $1 \leq s.length, t.length \leq 5 * 10^4$
- s 和 t 仅包含小写字母

① C++ • 智能模式 ② 模拟面试 | i ⌂ >_ ⌂ ⌂

```
1 class Solution {
2 public:
3     bool isAnagram(string s, string t) {
4         if (s.length() != t.length()) return false;
5         unordered_map<char, int> m1, m2;
6         for (int i = 0; i < s.length(); i++) {
7             m1[s[i]] += 1;
8             m2[t[i]] += 1;
9         }
10        if (m1 == m2) return true;
11        return false;
12    }
13};
```

• 383

383. 赎金信

难度 简单 723 ⭐ 🔍 🗑 📡

给你两个字符串： `ransomNote` 和 `magazine`，判断 `ransomNote` 能不能由 `magazine` 里面的字符构成。

如果可以，返回 `true`；否则返回 `false`。

`magazine` 中的每个字符只能在 `ransomNote` 中使用一次。

示例 1：

输入: `ransomNote = "a"`, `magazine = "b"`

输出: `false`

示例 2：

输入: `ransomNote = "aa"`, `magazine = "ab"`

输出: `false`

示例 3：

输入: `ransomNote = "aa"`, `magazine = "aab"`

输出: `true`

提示:

- $1 \leq \text{ransomNote.length}, \text{magazine.length} \leq 10^5$
- `ransomNote` 和 `magazine` 由小写英文字母组成

① C++ • 智能模式 ② 模拟面试 | i ⏪ ⏴ ⏵ ⏶ ⏷

```
1 class Solution {
2 public:
3     bool canConstruct(string ransomNote, string magazine) {
4         unordered_map<char, int> m;
5         for (int i = 0; i < ransomNote.length(); i++) {
6             m[ransomNote[i]] += 1;
7         }
8         for (int i = 0; i < magazine.length(); i++) {
9             if (m.count(magazine[i]) > 0 && (--m[magazine[i]] == 0)) {
10                 m.erase(magazine[i]);
11             }
12         }
13         if (m.size() == 0) return true;
14         return false;
15     }
16 }
```

49. 字母异位词分组

难度 中等 1489

给你一个字符串数组，请你将字母异位词组合在一起。可以按任意顺序返回结果列表。

字母异位词是由重新排列源单词的字母得到的一个新单词，所有源单词中的字母通常恰好只用一次。

示例 1：

输入: strs = ["eat", "tea", "tan", "ate", "nat", "bat"]

输出： [["bat"], ["nat", "tan"], ["ate", "eat", "tea"]]

示例 2：

输入: strs = [""]

输出： [[“”]]

示例 3：

输入: strs = ["a"]

输出： [["a"]]

提示：

- $1 \leq \text{strs.length} \leq 10^4$
 - $0 \leq \text{strs[i].length} \leq 100$

```
i C++ • 智能模式 模拟面试 | i P D > ⚙️ [ ]
```

```
1 class Solution {
2 public:
3     vector<vector<string>> groupAnagrams(vector<string>& strs) {
4         vector<vector<string>> result;
5         map<string, vector<string>> mm;
6         for (auto each : strs) {      定义一个string计数每个字符串
7             string temp(26, '0');
8             for (auto each_char : each) temp[each_char - 'a']++;
9             mm[temp].emplace_back(each);
10        }
11        for (auto each : mm) {
12            result.push_back(each.second);
13        }
14    }
15    return result;
16 }
```

```
class Solution {
public:
    vector<vector<string>> groupAnagrams(vector<string>& strs) {
        unordered_map<string, vector<string>> mp;
        for (string& str: strs) {
            string key = str;
            sort(key.begin(), key.end());
            mp[key].emplace_back(str);
        }
        vector<vector<string>> ans;      排序法
        for (auto it = mp.begin(); it != mp.end(); ++it) {
            ans.emplace_back(it->second);
        }
        return ans;
    }
};
```

- 438

438. 找到字符串中所有字母异位词

难度 中等 1173 收藏 分享 文章 提交

给定两个字符串 s 和 p ，找到 s 中所有 p 的 异位词 的子串，返回这些子串的起始索引。不考虑答案输出的顺序。

异位词 指由相同字母重排列形成的字符串（包括相同的字符串）。

示例 1:

输入: $s = "cbaebabacd"$, $p = "abc"$

输出: $[0,6]$

解释:

起始索引等于 0 的子串是 "cba"，它是 "abc" 的异位词。

起始索引等于 6 的子串是 "bac"，它是 "abc" 的异位词。

示例 2:

输入: $s = "abab"$, $p = "ab"$

输出: $[0,1,2]$

解释:

起始索引等于 0 的子串是 "ab"，它是 "ab" 的异位词。

起始索引等于 1 的子串是 "ba"，它是 "ab" 的异位词。

起始索引等于 2 的子串是 "ab"，它是 "ab" 的异位词。

提示:

- $1 \leq s.length, p.length \leq 3 * 10^4$
- s 和 p 仅包含小写字母

```
① C++ ② 智能模式 ③ 模拟面试 | i P ⌂ >_ ⚙ [ ]  
1 class Solution {  
2 public:  
3     vector<int> findAnagrams(string s, string p) {  
4         vector<int> result;  
5         unordered_map<char, int> ms, mp;  
6         int p_len = p.length(), s_len = s.length();  
7         for (int i = 0; i < p_len; i++) {  
8             ms[s[i]] += 1;  
9             mp[p[i]] += 1;  
10        }  
11        if (ms == mp) result.push_back(0);  
12        for (int i = p_len; i < s_len; i++) {  
13            ms[s[i]] += 1;  
14            if (--ms[s[i-p_len]] == 0) ms.erase(s[i-p_len]);  
15            if (ms == mp) result.push_back(i-p_len+1);  
16        }  
17        return result;  
18    }  
19};
```

固定长度为p_len的窗口

4.3 两个数组的交集 (349、350)

• 349

349. 两个数组的交集

难度 简单 山 766 ☆ ⌂ 文 ⌂ ⌂

给定两个数组 `nums1` 和 `nums2`，返回 它们的交集。输出结果中的每个元素一定是 唯一 的。我们可以 不考虑 输出结果的顺序。

示例 1：

输入: `nums1 = [1,2,2,1]`, `nums2 = [2,2]`
输出: `[2]`

示例 2：

输入: `nums1 = [4,9,5]`, `nums2 = [9,4,9,8,4]`
输出: `[9,4]`
解释: `[4,9]` 也是可通过的

提示：

- $1 \leq \text{nums1.length}, \text{nums2.length} \leq 1000$
- $0 \leq \text{nums1}[i], \text{nums2}[i] \leq 1000$

① C++ 智能模式 模拟面试 i P ⌂ > ⌂

```
1 class Solution {
2 public:
3     vector<int> intersection(vector<int>& nums1, vector<int>& nums2) {
4         int len1 = nums1.size();
5         int len2 = nums2.size();
6         unordered_set<int> s1, s2;
7         vector<int> result;
8         for (int i = 0; i < min(len1, len2); i++) {
9             s1.insert(nums1[i]);
10            s2.insert(nums2[i]);
11        }
12        if (len1 < len2) {
13            for (int i = len1; i < len2; i++) {
14                s2.insert(nums2[i]);
15            }
16        } else {
17            for (int i = len2; i < len1; i++) {
18                s1.insert(nums1[i]);
19            }
20        }
21        for (auto each : s1) {
22            if (s2.count(each) != 0) result.push_back(each);
23        }
24    }
25    return result;
26}
```



① C++ 智能模式 模拟面试 i P ⌂ > ⌂

```
1 class Solution {
2 public:
3     vector<int> intersection(vector<int>& nums1, vector<int>& nums2) {
4         vector<int> answer;
5         unordered_set<int> hash_set;
6         for(int i=0;i<nums1.size();i++){
7             if(hash_set.count(nums1[i])==0){
8                 hash_set.insert(nums1[i]);
9             }
10        }
11        for(int j =0;j<nums2.size();j++){
12            if(hash_set.count(nums2[j])>0){
13                answer.push_back(nums2[j]);
14                hash_set.erase(nums2[j]);
15            }
16        }
17    }
18    return answer;
19}
```

每个数组只分别遍历一次的方法

```

1 class Solution {
2 public:
3     vector<int> intersection(vector<int>& nums1, vector<int>& nums2) {
4         unordered_set<int> result_set; // 存放结果, 之所以用set是为了给结果集去重
5         unordered_set<int> nums_set(nums1.begin(), nums1.end());
6         for (int num : nums2) {
7             // 发现nums2的元素 在nums_set里又出现过
8             if (nums_set.find(num) != nums_set.end()) {
9                 result_set.insert(num);
10            }
11        }
12        return vector<int>(result_set.begin(), result_set.end());
13    }
14 };

```

- 时间复杂度: $O(mn)$
- 空间复杂度: $O(n)$

代码随想录版本：注意vector的赋值（根据set）

• 350

350. 两个数组的交集 II

难度 简单 969 ⭐ ⏴ 文档 ⌂

给你两个整数数组 `nums1` 和 `nums2`，请你以数组形式返回两数组的交集。返回结果中每个元素出现的次数，应与元素在两个数组中都出现的次数一致（如果出现次数不一致，则考虑取较小值）。可以不考虑输出结果的顺序。

示例 1:

输入: `nums1 = [1,2,2,1]`, `nums2 = [2,2]`
 输出: `[2,2]`

示例 2:

输入: `nums1 = [4,9,5]`, `nums2 = [9,4,9,8,4]`
 输出: `[4,9]`

提示:

- `1 <= nums1.length, nums2.length <= 1000`
- `0 <= nums1[i], nums2[i] <= 1000`

● 智能模式

笨办法：分别遍历，再遍历哈希映射

```
1 class Solution {
2 public:
3     vector<int> intersect(vector<int>& nums1, vector<int>& nums2) {
4         int len1 = nums1.size();
5         int len2 = nums2.size();
6         unordered_map<int, int> m1, m2;
7         vector<int> result;
8         for (int i = 0; i < min(len1, len2); i++) {
9             m1[nums1[i]] += 1;
10            m2[nums2[i]] += 1;
11        }
12        if (len1 < len2) {
13            for (int i = len1; i < len2; i++) {
14                m2[nums2[i]] += 1;
15            }
16        } else {
17            for (int i = len2; i < len1; i++) {
18                m1[nums1[i]] += 1;
19            }
20        }
21        for (auto each : m1) {
22            if (m2.count(each.first) != 0) {
23                vector<int> temp(min(each.second, m2[each.first]), each.first);
24                result.insert(result.end(), temp.begin(), temp.end());
25            }
26        }
27        return result;
28    }
}
```

注意此处合并两个vector的代码

只遍历每个数组各一遍的方法

```
class Solution {
public:
    vector<int> intersect(vector<int>& nums1, vector<int>& nums2) {
        vector<int> answer;
        unordered_map<int, int> hashmap;
        for(int i=0;i<nums1.size();i++){
            if(hashmap.count(nums1[i])==0) hashmap.insert(make_pair(nums1[i], 1));
            else hashmap[nums1[i]] += 1;
        }
        for(int j=0;j<nums2.size();j++){
            if(hashmap.count(nums2[j])>0 && hashmap[nums2[j]]>0){
                answer.push_back(nums2[j]);
                hashmap[nums2[j]] -= 1;
            }
        }
        return answer;
    }
};
```

4.4 快乐数 (202)

- 202

202. 快乐数

难度 简单 1311 收藏 分享

编写一个算法来判断一个数 n 是不是快乐数。

「快乐数」定义为：

- 对于一个正整数，每一次将该数替换为它每个位置上的数字的平方和。
- 然后重复这个过程直到这个数变为 1，也可能是 无限循环 但始终变不到 1。
- 如果这个过程 结果为 1，那么这个数就是快乐数。

如果 n 是 快乐数 就返回 `true`；不是，则返回 `false`。

示例 1：

输入： $n = 19$

输出： `true`

解释：

$$1^2 + 9^2 = 82$$

$$8^2 + 2^2 = 68$$

$$6^2 + 8^2 = 100$$

$$1^2 + 0^2 + 0^2 = 1$$

示例 2：

输入： $n = 2$

输出： `false`

提示：

- $1 \leq n \leq 2^{31} - 1$

```
① C++ ② 智能模式 ③ 模拟面试 | i P ⌂ >_ ⚙ []  
1 class Solution {  
2     public:  
3         bool isHappy(int n) {  
4             unordered_set<int> s;  
5             int sums = n, temp;  
6             while (sums != 1) { 平方和=1, 跳出循环, 是快乐数  
7                 temp = sums;  
8                 sums = 0;  
9                 while (temp) {  
10                     sums += pow((temp % 10), 2); 找得到, 陷入循环  
11                     temp /= 10;  
12                 }  
13                 if (s.find(sums) != s.end()) return false; 找不到, 先加入哈希集  
14                 else s.insert(sums);  
15             }  
16             return true;  
17         }  
18     };
```

4.5 两数之和 (1——梦开始的地方)

- 1

1. 两数之和

难度 简单 17041 ⭐ 🔍 💾 🗑

给定一个整数数组 `nums` 和一个整数目标值 `target`，请你在该数组中找出 **和为目标值 `target`** 的那 **两个** 整数，并返回它们的数组下标。

你可以假设每种输入只会对应一个答案。但是，数组中同一个元素在答案里不能重复出现。

你可以按任意顺序返回答案。

示例 1：

输入: `nums = [2,7,11,15]`, `target = 9`

输出: `[0,1]`

解释: 因为 `nums[0] + nums[1] == 9`，返回 `[0, 1]`。

示例 2：

输入: `nums = [3,2,4]`, `target = 6`

输出: `[1,2]`

示例 3：

输入: `nums = [3,3]`, `target = 6`

输出: `[0,1]`

提示:

- $2 \leq \text{nums.length} \leq 10^4$
- $-10^9 \leq \text{nums}[i] \leq 10^9$
- $-10^9 \leq \text{target} \leq 10^9$
- 只会存在一个有效答案

```
① C++ • 智能模式 ② 模拟面试 | i P ⌂ >_ ⚙ [ ]  
1 class Solution {  
2 public:  
3     vector<int> twoSum(vector<int>& nums, int target) {  
4         vector<int> result;  
5         map<int, int> m;  
6         for (int i = 0; i < nums.size(); i++) {  
7             if (m.find(target - nums[i]) != m.end()) {  
8                 result.push_back(m[target - nums[i]]);  
9                 result.push_back(i);  
10                return result;  
11            } else {  
12                m[nums[i]] = i;  
13            }  
14        }  
15        return result;  
16    }  
17};
```

4.6 四数之和 (454)

- 454

454. 四数相加 II

难度 中等 824 收藏 分享 文档

给你四个整数数组 `nums1`、`nums2`、`nums3` 和 `nums4`，数组长度都是 `n`，请你计算有多少个元组 (i, j, k, l) 能满足：

- $0 \leq i, j, k, l < n$
- $\text{nums1}[i] + \text{nums2}[j] + \text{nums3}[k] + \text{nums4}[l] == 0$

示例 1：

输入: `nums1 = [1,2]`, `nums2 = [-2,-1]`, `nums3 = [-1,2]`, `nums4 = [0,2]`

输出: 2

解释:

两个元组如下：

1. $(0, 0, 0, 1) \rightarrow \text{nums1}[0] + \text{nums2}[0] + \text{nums3}[0] + \text{nums4}[1] = 1 + (-2) + (-1) + 2 = 0$
2. $(1, 1, 0, 0) \rightarrow \text{nums1}[1] + \text{nums2}[1] + \text{nums3}[0] + \text{nums4}[0] = 2 + (-1) + (-1) + 0 = 0$

示例 2：

输入: `nums1 = [0]`, `nums2 = [0]`, `nums3 = [0]`, `nums4 = [0]`

输出: 1

提示:

- $n == \text{nums1.length}$
- $n == \text{nums2.length}$
- $n == \text{nums3.length}$
- $n == \text{nums4.length}$
- $1 \leq n \leq 200$
- $-2^{28} \leq \text{nums1}[i], \text{nums2}[i], \text{nums3}[i], \text{nums4}[i] \leq 2^{28}$

```
① C++ ▾ • 智能模式 模拟面试 i P ⌂ > ⚙ [·]
1 class Solution {
2 public:
3     int fourSumCount(vector<int>& nums1, vector<int>& nums2, vector<int>& nums3,
4                      vector<int>& nums4) {
5         int nums_size = nums1.size();
6         unordered_map<int, int> m;
7         int result = 0;
8         for(int i = 0; i < nums_size; i++) {
9             for (int j = 0; j < nums_size; j++) {
10                 m[-nums1[i]-nums2[j]] += 1;
11             }
12         }
13         for(int i = 0; i < nums_size; i++) {
14             for (int j = 0; j < nums_size; j++) {
15                 if (m.find(nums3[i]+nums4[j]) != m.end()) result += m[nums3[i]+nums4
16 [j]];
17             }
18         }
19     }
20 }
```

4.7 三数之和 (15——非哈希表解法，哈希表很难写)

- ### • 15 (双指针法)

15. 三数之和

难度 中等 山 5954 ★ ⌂ 🔍 🔔

给你一个整数数组 `nums`，判断是否存在三元组 `[nums[i], nums[j], nums[k]]` 满足 `i != j`、`i != k` 且 `j != k`，同时还满足 `nums[i] + nums[j] + nums[k] == 0`。请

你返回所有和为 `0` 且不重复的三元组。

注意：答案中不可以包含重复的三元组。

示例 1：

输入: `nums = [-1,0,1,2,-1,-4]`

输出: `[[-1,-1,2],[-1,0,1]]`

解释:

`nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0`。

`nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0`。

`nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0`。

不同的三元组是 `[-1,0,1]` 和 `[-1,-1,2]`。

注意，输出的顺序和三元组的顺序并不重要。

示例 2：

输入: `nums = [0,1,1]`

输出: `[]`

解释: 唯一可能的三元组和不为 `0`。

示例 3：

输入: `nums = [0,0,0]`

输出: `[[0,0,0]]`

解释: 唯一可能的三元组和为 `0`。

提示：

- `3 <= nums.length <= 3000`
- `-105 <= nums[i] <= 105`

```
class Solution {
public:
    vector<vector<int>> threeSum(vector<int>& nums) {
        vector<vector<int>> answer;
        if(nums.size() < 3) return answer;
```

```

sort(nums.begin(), nums.end());
vector<int> temp;
int size = nums.size();
for(int i=0;i<size;i++){
    if(nums[i] > 0) return answer;
    if(i > 0 && nums[i] == nums[i-1]) continue; //保证无重复，因为若i和i-1相等，则
i-1遍历时就已经把可能的答案装进answer了，也即若i和i-1相等，i-1的所有可能答案包含i的所有可能答案，故跳
过，所以直接continue i++
    int left = i+1;
    int right = size-1;
    while(left < right){
        if(nums[i] + nums[left] + nums[right] == 0){
            temp.push_back(nums[i]);
            temp.push_back(nums[left]);
            temp.push_back(nums[right]);
            answer.push_back(temp);
            temp.clear();
            while(left < right && nums[left] == nums[left+1]) left++;
            while(left < right && nums[right] == nums[right-1]) right--;
            left++;
            right--;
        }
        else if(nums[i] + nums[left] + nums[right] > 0) right--;
        else left++;
    }
}
return answer;
}

```

4.8 四数之和（18——非哈希表解法，哈希表很难写）

- 18（双指针法）

18. 四数之和

难度 中等 1600 收藏 分享 提交

给你一个由 n 个整数组成的数组 nums ，和一个目标值 target 。请你找出并返回满足下述全部条件且不重复的四元组 $[\text{nums}[a], \text{nums}[b], \text{nums}[c], \text{nums}[d]]$ （若两个四元组元素一一对应，则认为两个四元组重复）：

- $0 \leq a, b, c, d < n$
- a 、 b 、 c 和 d 互不相同
- $\text{nums}[a] + \text{nums}[b] + \text{nums}[c] + \text{nums}[d] == \text{target}$

你可以按任意顺序 返回答案。

示例 1:

输入: $\text{nums} = [1,0,-1,0,-2,2]$, $\text{target} = 0$
输出: $[[[-2,-1,1,2],[-2,0,0,2],[-1,0,0,1]]$

示例 2:

输入: $\text{nums} = [2,2,2,2,2]$, $\text{target} = 8$
输出: $[[2,2,2,2]]$

提示:

- $1 \leq \text{nums.length} \leq 200$
- $-10^9 \leq \text{nums}[i] \leq 10^9$
- $-10^9 \leq \text{target} \leq 10^9$

```
class Solution {
public:
    vector<vector<int>> fourSum(vector<int>& nums, int target) {
        vector<vector<int>> answer;
        if(nums.size() < 4) return answer;
        sort(nums.begin(), nums.end());
        int a, b, c, d;
        int num_size = nums.size();
        for(a = 0; a < num_size - 3; a++){
            if(a > 0 && nums[a] == nums[a-1]) continue;
            for(b = a+1; b < num_size - 2; b++){
                if(b > a+1 && nums[b] == nums[b-1]) continue;
                c = b+1;
                d = num_size - 1;
                while(c < d){
                    if((long) nums[a] + nums[b] + nums[c] + nums[d] == target){
                        answer.push_back({nums[a], nums[b], nums[c], nums[d]});
                    }
                    c++;
                    d--;
                }
            }
        }
        return answer;
    }
};
```

```

        answer.push_back(vector<int>{nums[a], nums[b], nums[c],
nums[d]});

        while(c < d && nums[c] == nums[c+1]) c++;
        while(c < d && nums[d] == nums[d-1]) d--;
        c++;
        d--;
    }

    else if((long) nums[a] + nums[b] + nums[c] + nums[d] < target){
        c++;
    }
    else{
        d--;
    }
}
}

return answer;
}
};


```

五、字符串

5.1 反转字符串 (344)

- 344

344. 反转字符串

难度 简单 775 ☆ ⚡ ⚡ ⚡

编写一个函数，其作用是将输入的字符串反转过来。输入字符串以字符数组 `s` 的形式给出。

不要给另外的数组分配额外的空间，你必须原地修改输入数组、使用 O(1) 的额外空间解决这一问题。

示例 1：

输入: `s = ["h", "e", "l", "l", "o"]`

输出: `["o", "l", "l", "e", "h"]`

示例 2：

输入: `s = ["H", "a", "n", "n", "a", "h"]`

输出: `["h", "a", "n", "n", "a", "H"]`

提示：

- `1 <= s.length <= 105`
- `s[i]` 都是 ASCII 码表中的可打印字符

```
i C++ • 智能模式  
④ 模拟面试 | i P D > ⚙ [ ]  
1 class Solution {  
2     public:  
3         void reverseString(vector<char>& s) {  
4             int s_size = s.size();  
5             if (s_size == 1) return;  
6             for (int i = 0, j = s_size-1; i < j; i++, j--) {  
7                 char temp = s[i];  
8                 s[i] = s[j];  
9                 s[j] = temp;  
10            }  
11        return;  
12    }  
13 };
```

5.2 反转字符串 2 (541)

- 541

541. 反转字符串 II

难度 简单 山 470 ⭐ ⌂ Ⓛ 🔍

给定一个字符串 `s` 和一个整数 `k`，从字符串开头算起，每计数至 `2k` 个字符，就反转这 `2k` 字符中的前 `k` 个字符。

- 如果剩余字符少于 `k` 个，则将剩余字符全部反转。
- 如果剩余字符小于 `2k` 但大于或等于 `k` 个，则反转前 `k` 个字符，其余字符保持原样。

示例 1：

输入: `s = "abcdefg", k = 2`
输出: "bacdfeg"

示例 2：

输入: `s = "abcd", k = 2`
输出: "bacd"

提示：

- `1 <= s.length <= 104`
- `s` 仅由小写英文组成
- `1 <= k <= 104`

```
① C++ 智能模式 模拟面试 i P ⌂ >_ ⚙ [ ]  
1 class Solution {  
2     public:  
3         string reverseStr(string s, int k) {  
4             for (int i = 0; i < s.size(); i += (2*k)) {  
5                 if ((s.size() - i) >= k) {  
6                     reverse(s, i, i + k - 1);  
7                 } else {  
8                     reverse(s, i, s.size() - 1);  
9                 }  
10            }  
11            return s;  
12        }  
13  
14        void reverse(string& s, int start, int end) {  
15            for (int i = start, j = end; i < j; i++, j--) {  
16                swap(s[i], s[j]);  
17            }  
18        }  
19    };
```

5.3 替换空格 (剑指offer.05)

- 剑指offer.05

剑指 Offer 05. 替换空格

难度 简单 490 收藏 分享

请实现一个函数，把字符串 `s` 中的每个空格替换成"%20"。

示例 1：

输入: `s = "We are happy."`

输出: `"We%20are%20happy."`

限制:

`0 <= s 的长度 <= 10000`

• 智能模式

双指针法

```
1 class Solution {
2 public:
3     string replaceSpace(string s) {
4         int num_space = count(s.begin(), s.end(), ' '); // 统计空格个数, count
5         int j = s.size() - 1, i = j + num_space * 2; // j指向旧字符串尾, i指向新字符串尾
6         s.resize(i + 1);
7         while (i != j) { // ij相等时说明空格已经替换完成
8             if (s[j] != ' ') {
9                 s[i] = s[j];
10                i--;
11                j--;
12            } else {
13                s[i--] = '0';
14                s[i--] = '2'; // s[j] == ' ', 则s[i-2],s[i-1],s[i]赋值'%20', ij向前移一位
15                s[i--] = '%';
16                j--;
17            }
18        }
19        return s;
20    }
21};
```

5.4 反转字符串中的单词（151）

• 151

151. 反转字符串中的单词

难度 中等 878 收藏 文档 提交

给你一个字符串 `s`，请你反转字符串中 单词 的顺序。

单词 是由非空格字符组成的字符串。`s` 中使用至少一个空格将字符串中的 单词 分隔开。

返回 单词 顺序颠倒且 单词 之间用单个空格连接的结果字符串。

注意：输入字符串 `s` 中可能会存在前导空格、尾随空格或者单词间的多个空格。返回的结果字符串中，单词间应当仅用单个空格分隔，且不包含任何额外的空格。

示例 1：

输入: `s = "the sky is blue"`

输出: `"blue is sky the"`

示例 2：

输入: `s = " hello world "`

输出: `"world hello"`

解释: 反转后的字符串中不能存在前导空格和尾随空格。

示例 3：

输入: `s = "a good example"`

输出: `"example good a"`

解释: 如果两个单词间有多余的空格，反转后的字符串需要将单词间的空格减少到仅有一个。

提示:

- `1 <= s.length <= 104`
- `s` 包含英文大小写字母、数字和空格
- `s` 中 至少存在一个 单词

```
① C++ • 智能模式 ② 模拟面试 | i P ⌂ > ⌂ ⌂
```

```
1 class Solution {
2 public:
3     string reverseWords(string s) {
4         int i = 0, j = 0;
5         while (j < s.size() && s[j] == ' ') j++;
6         for (; j < s.size(); ) {
7             if (s[j] == ' ' && s[j] == s[j-1]) j++;
8             else {
9                 s[i++] = s[j++];
10            }
11        }
12        if (s[i-1] == ' ') s.resize(i-1);
13        else s.resize(i);           // 如果最后有空格，则resize(i-1)，否则resize(i)
14
15        reverse(s, 0, s.size()-1);   // 反转整个字符串
16        for (i = 0, j = 0; j < s.size(); ) {
17            while (j < s.size() && s[j] != ' ') j++;
18            reverse(s, i, j-1);       // 反转单个小字符串
19            i = ++j;
20        }
21        return s;
22    }
23
24    void reverse(string& s, int start, int end) {
25        char temp;
26        while (start < end) {
27            swap(s[start++], s[end--]);
28        }
29    }
30}
```

5.5 左旋转字符串（剑指Offer58-II）

- 剑指Offer58-II

剑指 Offer 58 - II. 左旋转字符串

难度 简单 403 收藏 分享 报错

字符串的左旋转操作是把字符串前面的若干个字符转移到字符串的尾部。请定义一个函数实现字符串左旋转操作的功能。比如，输入字符串"abcdefg"和数字2，该函数将返回左旋转两位得到的结果"cdefgab"。

示例 1：

输入: s = "abcdefg", k = 2

输出: "cdefgab"

示例 2：

输入: s = "lrloseumgh", k = 6

输出: "umghlrlose"

限制：

- $1 \leq k < s.length \leq 10000$

```
(i) C++ • 智能模式 i P ⌂ >_ ⚙ []  
1 class Solution {  
2 public:  
3     string reverseLeftWords(string s, int n) {  
4         reverse(s, 0, n-1);  
5         reverse(s, n, s.size()-1);  
6         reverse(s, 0, s.size()-1);  
7         return s;  
8     }  
9  
10    void reverse(string& s, int start, int end) {  
11        while (start < end) swap(s[start++], s[end--]);  
12    }  
13};
```

5.6 找出字符串第一个匹配项的下标 (28)

- 28

28. 找出字符串中第一个匹配项的下标

难度 中等 1875 收藏 提交 资源

给你两个字符串 `haystack` 和 `needle`，请你在 `haystack` 字符串中找出 `needle` 字符串的第一个匹配项的下标（下标从 0 开始）。如果 `needle` 不是 `haystack` 的一部分，则返回 `-1`。

示例 1：

输入: `haystack = "sadbutsad", needle = "sad"`

输出: `0`

解释: "sad" 在下标 `0` 和 `6` 处匹配。

第一个匹配项的下标是 `0`，所以返回 `0`。

示例 2：

输入: `haystack = "leetcode", needle = "leeto"`

输出: `-1`

解释: "leeto" 没有在 "leetcode" 中出现，所以返回 `-1`。

提示：

- $1 \leq \text{haystack.length}, \text{needle.length} \leq 10^4$
- `haystack` 和 `needle` 仅由小写英文字母组成

KMP算法，代码随想录版本

```

1 class Solution {
2 public:
3     int strStr(string haystack, string needle) {
4         if (needle.size() == 0) return 0;
5         vector<int> next = buildNext(needle);
6         int n = haystack.size(), i = 0;
7         int m = needle.size(), j = 0;
8         while (i < n && j < m) {
9             if (haystack[i] == needle[j]) i++, j++;
10            else if (j != 0) j = next[j-1];
11            else i++;
12        }
13        if (j == m) return i - j;
14        return -1;
15    }
16
17    vector<int> buildNext(string needle) {
18        int size = needle.size();
19        int j = 0;                                1. 初始化
20        vector<int> next(size, 0);
21        next[j] = 0;
22        for (int i = 1; i < size; i++) {          2. 前后缀不匹配, j回退
23            while (j > 0 && needle[j] != needle[i]) j = next[j-1];
24            if (needle[j] == needle[i]) j++;      3. 前后缀匹配, j++
25            next[i] = j;                         4. 更新next
26        }
27        return next;
28    }

```

例：aabaaf的next
010120

*i*含义：后缀字符串末尾
*j*含义：前缀字符串末尾 and
j位及之前位的最大前后缀长度

n、动态规划 (dp)

.1 动态规划基础



动态规划

背包问题

完全背包

- 01背包理论基础
- 01背包理论基础 (滚动数组)

0416. 分割等和子集

1049. 最后一块石头的重量II

0494. 目标和

0474. 一和零

完全背包理论基础

0518. 零钱兑换II

0377. 组合总和IV

0070. 爬楼梯 (完全背包解法)

0322. 零钱兑换

0279. 完全平方数

0139. 单词拆分

多重背包理论基础

力扣暂时没发现对应题目

背包问题大总结

打家劫舍

198. 打家劫舍

213. 打家劫舍II

337. 打家劫舍III

股票问题

121. 买卖股票的最佳时机 (只能买卖一次)

122. 买卖股票的最佳时机II (可以买卖多次)

123. 买卖股票的最佳时机III (最多买卖两次)

188. 买卖股票的最佳时机IV (最多买卖k次)

309. 最佳买卖股票时机含冷冻期 (买卖多次 , 卖出有一天冷冻期)

714. 买卖股票的最佳时机含手续费 (买卖多次 , 每次有手续费)

子序列 (不连续)

300. 最长上升子序列

1143. 最长公共子序列

1035. 不相交的线

674. 最长连续递增序列

子序列 (连续)

718. 最长重复子数组



对于动态规划问题，我将拆解为如下五步曲，这五步都搞清楚了，才能说把动态规划真的掌握了！

1. 确定dp数组（dp table）以及下标的含义
2. 确定递推公式
3. dp数组如何初始化
4. 确定遍历顺序
5. 举例推导dp数组

dp五部曲

.2 斐波那契数（509）

• 509

509. 斐波那契数

难度 简单 642 收藏 分享

斐波那契数（通常用 $F(n)$ 表示）形成的序列称为 **斐波那契数列**。该数列由 0 和 1 开始，后面的每一项数字都是前面两项数字的和。也就是：

$$\begin{aligned} F(0) &= 0, \quad F(1) = 1 \\ F(n) &= F(n - 1) + F(n - 2), \text{ 其中 } n > 1 \end{aligned}$$

给定 n ，请计算 $F(n)$ 。

示例 1：

输入: $n = 2$
 输出: 1
 解释: $F(2) = F(1) + F(0) = 1 + 0 = 1$

```
1 class Solution {
2 public:
3     int fib(int n) {
4         // 动态规划写法
5         if (n <= 1) return n;
6         vector<int> dp(n+1);
7         dp[0] = 0, dp[1] = 1;
8         for (int i = 2; i <= n; i++) {
9             dp[i] = dp[i-1] + dp[i-2];
10            cout << dp[i] << endl;
11        }
12        return dp[n];
13    }
14 };
15
16 class Solution {
17 public:
18     int fib(int n) {
19         // 三变量简洁写法
20         if (n <= 1) return n;
21         int dp0 = 0, dp1 = 1;
22         int result;
23         for (int i = 2; i <= n; i++) {
24             result = dp0 + dp1;
25             dp0 = dp1;
26             dp1 = result;
27         }
28         return result;
29     }
}
```



.3 爬楼梯 (70)

- 70

70. 爬楼梯

难度 简单 3033 收藏 举报

假设你正在爬楼梯。需要 n 阶你才能到达楼顶。

每次你可以爬 1 或 2 个台阶。你有多少种不同的方法可以爬到楼顶呢？

示例 1：

输入: $n = 2$

输出: 2

解释: 有两种方法可以爬到楼顶。

1. 1 阶 + 1 阶

2. 2 阶

示例 2：

输入: $n = 3$

输出: 3

解释: 有三种方法可以爬到楼顶。

1. 1 阶 + 1 阶 + 1 阶

2. 1 阶 + 2 阶

3. 2 阶 + 1 阶

```
① C++ ② 智能模式 ③ 模拟面试 | i P ⌂ >_ ⚙ [ ]
```

```
1 class Solution {
2 public:
3     int climbStairs(int n) {
4         if (n <= 2) return n;
5         vector<int> dp(n);
6         dp[0] = 1, dp[1] = 2;
7         for (int i = 2; i < n; i++) {
8             dp[i] = dp[i-1] + dp[i - 2];
9         }
10        return dp[n-1];
11    }
12 }
```

.4 使用最小花费爬楼梯 (746)

- 746

746. 使用最小花费爬楼梯

难度 简单 山 1213 ☆ ⚡ 文 A 🔍

给你一个整数数组 `cost`，其中 `cost[i]` 是从楼梯第 `i` 个台阶向上爬需要支付的费用。一旦你支付此费用，即可选择向上爬一个或者两个台阶。

你可以选择从下标为 `0` 或下标为 `1` 的台阶开始爬楼梯。

请你计算并返回达到楼梯顶部的最低花费。

示例 1：

输入: `cost = [10, 15, 20]`

输出: 15

解释: 你将从下标为 1 的台阶开始。

- 支付 15，向上爬两个台阶，到达楼梯顶部。

总花费为 15。

示例 2：

输入: `cost = [1, 100, 1, 1, 1, 100, 1, 1, 100, 1]`

输出: 6

解释: 你将从下标为 0 的台阶开始。

- 支付 1，向上爬两个台阶，到达下标为 2 的台阶。

- 支付 1，向上爬两个台阶，到达下标为 4 的台阶。

- 支付 1，向上爬两个台阶，到达下标为 6 的台阶。

- 支付 1，向上爬一个台阶，到达下标为 7 的台阶。

- 支付 1，向上爬两个台阶，到达下标为 9 的台阶。

- 支付 1，向上爬一个台阶，到达楼梯顶部。

总花费为 6。

提示：

- `2 <= cost.length <= 1000`
- `0 <= cost[i] <= 999`

```
1 class Solution {
2 public:
3     int minCostClimbingStairs(vector<int>& cost) {
4         int cost_size = cost.size();
5         if(cost_size == 2) return min(cost[0], cost[1]);
6         vector<int> dp(cost_size);
7         dp[0] = cost[0], dp[1] = cost[1];
8         for (int i = 2; i < cost_size; i++) {
9             dp[i] = cost[i] + min(dp[i-2], dp[i-1]);
10        }
11        return min(dp[cost_size-2], dp[cost_size-1]);
12    }
13};
```

.5 不同路径 (62、63)

• 62

62. 不同路径

难度 中等 1763

一个机器人位于一个 $m \times n$ 网格的左上角（起始点在下图中标记为“Start”）。

机器人每次只能向下或者向右移动一步。机器人试图达到网格的右下角（在下图中标记为“Finish”）。

问总共有多少条不同的路径?

示例 1：



输入: $m = 3, n = 7$

输出：28

示例 2:

输入: m = 3, n = 2

输出: 3

解释:

从左上角开始, 总共有 3 条路径可以到达右下角。

1. 向右 -> 向下 -> 向下
2. 向下 -> 向下 -> 向右
3. 向下 -> 向右 -> 向下

示例 3:

输入: m = 7, n = 3

输出: 28

示例 4:

输入: m = 3, n = 3

输出: 6

提示:

- $1 \leq m, n \leq 100$
- 题目数据保证答案小于等于 $2 * 10^9$

```
1 class Solution {
2 public:
3     int uniquePaths(int m, int n) {
4         vector<vector<int>> dp(m, vector<int>(n, 0));
5         for (int i = 0; i < m; i++) dp[i][0] = 1;
6         for (int j = 0; j < n; j++) dp[0][j] = 1;      第一列和第一行都初始化为1
7         for (int i = 1; i < m; i++) {
8             for (int j = 1; j < n; j++) {
9                 dp[i][j] = dp[i - 1][j] + dp[i][j - 1];
10            }
11        }
12        return dp[m - 1][n - 1];
13    }
14 }
```

63. 不同路径 II

难度 中等 1047

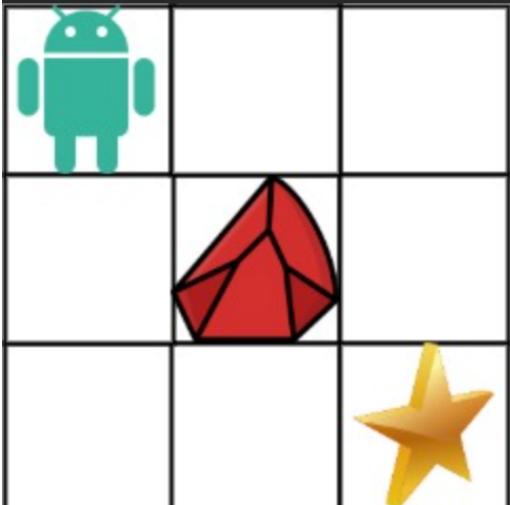
一个机器人位于一个 $m \times n$ 网格的左上角（起始点在下图中标记为“Start”）。

机器人每次只能向下或者向右移动一步。机器人试图达到网格的右下角（在下图中标记为“Finish”）。

现在考虑网格中有障碍物。那么从左上角到右下角将会有多少条不同的路径？

网格中的障碍物和空位置分别用 `1` 和 `0` 来表示。

示例 1：



输入: `obstacleGrid = [[0,0,0],[0,1,0],[0,0,0]]`

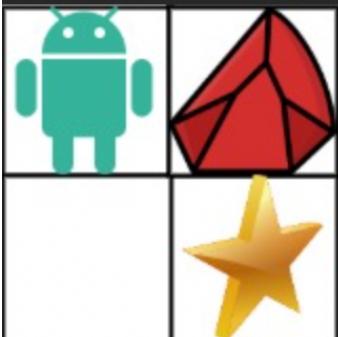
输出: 2

解释: 3x3 网格的正中间有一个障碍物。

从左上角到右下角一共有 2 条不同的路径:

1. 向右 -> 向右 -> 向下 -> 向下
2. 向下 -> 向下 -> 向右 -> 向右

示例 2：



输入: `obstacleGrid = [[0,1],[0,0]]`

输出: 1

提示:

- $m == \text{obstacleGrid.length}$
- $n == \text{obstacleGrid}[i].length$
- $1 \leq m, n \leq 100$
- $\text{obstacleGrid}[i][j]$ 为 0 或 1

① C++ • 智能模式 ② 模拟面试 | i P C >_ ⚙ []

```
1 class Solution {
2 public:
3     int uniquePathsWithObstacles(vector<vector<int>>& obstacleGrid) {
4         int m = obstacleGrid.size(), n = obstacleGrid[0].size();
5         vector<vector<int>> dp(m, vector<int>(n));
6         bool row_flag = false, col_flag = false;    先赋值第一行和第一列
7         // 第一行赋值, 如果有障碍物, 之后所有均0
8         for (int i = 0; i < n; i++) {
9             if (row_flag || obstacleGrid[0][i] == 1) {
10                 dp[0][i] = 0;
11                 row_flag = true;        第一行有障碍物, 障碍物之后的都是0
12             }
13             else dp[0][i] = 1;
14         }
15         // 第一列赋值, 如果有障碍物, 之后所有均0
16         for (int i = 0; i < m; i++) {
17             if (col_flag || obstacleGrid[i][0] == 1) {
18                 dp[i][0] = 0;
19                 col_flag = true;        第一列有障碍物, 障碍物之后的都是0
20             }
21             else dp[i][0] = 1;
22         }
23         for (int i = 1; i < m; i++) {
24             for (int j = 1; j < n; j++) {
25                 if (obstacleGrid[i][j] == 1) dp[i][j] = 0;
26                 else dp[i][j] = dp[i-1][j] + dp[i][j-1];    按顺序赋值即可
27             }
28         }
29     return dp[m-1][n-1];
30 }
```

.6 整数拆分 (343)

- 343

343. 整数拆分

难度 中等

1185



文



给定一个正整数 n ，将其拆分为 k 个 正整数 的和 ($k \geq 2$)，并使这些整数的乘积最大化。

返回 你可以获得的最大乘积。

示例 1:

输入: $n = 2$

输出: 1

解释: $2 = 1 + 1$, $1 \times 1 = 1$ 。

示例 2:

输入: $n = 10$

输出: 36

解释: $10 = 3 + 3 + 4$, $3 \times 3 \times 4 = 36$ 。

提示:

- $2 \leq n \leq 58$

C++ 智能模式 模拟面试

```

1 class Solution {
2 public:
3     int integerBreak(int n) {
4         vector<int> dp(n+1);
5         dp[0] = 0, dp[1] = 0, dp[2] = 1;
6         for (int i = 3; i <= n; i++) {
7             for (int j = 1; j <= i/2; j++) {
8                 dp[i] = max(j*(i-j), max(j*dp[i-j], dp[i]));
9             }
10        } 只拆分i-j不拆分j的原因，即为什么不写成dp[j]*dp[i-j]: 因为
11        return dp[n]; 拆分j的情况已经在之前被遍历到了，比如求dp[6]，拆分为15
12    } 和24，求2*dp[4]时不求dp[2]*dp[4]因为dp[2]已经在求1*dp[5]
13 }

```

1. dp[i]含义：分拆数字i，可以得到的最大乘积为dp[i]
2. 递推公式： $dp[i] = \max(j*(i-j), j*dp[i-j], dp[i])$
3. 初始化：dp[0]和dp[1]没有意义，初始化为0，对最终结果也没有影响
4. 遍历顺序：从前往后

将i拆分为j和i-j，则 $dp[i] = \max(j*(i-j), j*dp[i-j], dp[i])$ ，此处加上dp[i]是因为遍历j的过程中需要不断更新dp[i]，所以自己和自己max

.7 0-1背包问题（416）

0-1背包问题题目描述：有n件物品和一个最多能背重量为w的背包。第i件物品的重量是weight[i]，得到的价值是value[i]。每件物品只能用一次，求解将哪些物品装入背包里物品价值总和最大。假设物品信息如下，背包最多可装重量为4的物品。

	重量	价值
物品0	1	15
物品1	3	20
物品2	4	30

二维数据写法：

- dp [i] [j] 含义：表示从下标为 [0-i] 的物品里任意取，放进容量为j的背包，价值总和最大是多少
- 递推公式： $dp[i][j] = \max(dp[i-1][j], dp[i-1][j - weight[i]] + value[i])$ ，即不放物品i和放物品i
- 初始化： $dp[0][0] = 0$ ，因为背包重量为0时放不进东西。 $dp[0][j]$ 当j大于等于物品0的重量时， $dp[0][j] = value[0]$ ， $dp[0][j]$ 当j小于物品0的重量时， $dp[0][j] = 0$ 。其余位置均初始化为0即可（ $dp[i][j]$ 是由左上方数值推导出来了，那么其他下标初始为什么数值都可以，因为都会被覆盖）
- 遍历顺序：先物品后重量（更好） / 先重量后物品，都是靠左上方数据推出来的，所以先后不影响

```

void test_2_wei_bag_problem() {
    vector<int> weight = {1, 3, 4};
}

```

```

vector<int> value = {15, 20, 30};
int bagweight = 4;

// 二维数组
vector<vector<int>> dp(weight.size(), vector<int>(bagweight + 1, 0));

// 初始化
for (int j = weight[0]; j <= bagweight; j++) {
    dp[0][j] = value[0];
}

// weight数组的大小 就是物品个数
for(int i = 1; i < weight.size(); i++) { // 遍历物品
    for(int j = 0; j <= bagweight; j++) { // 遍历背包容量
        if (j < weight[i]) dp[i][j] = dp[i - 1][j];
        else dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - weight[i]] + value[i]);
    }
}

cout << dp[weight.size() - 1][bagweight] << endl;
}

int main() {
    test_2_wei_bag_problem1();
}

```

一维数据写法（滚动数组，自己覆盖自己）：

- $dp[j]$ 含义：容量为 j 的背包，所背的物品价值可以最大为 $dp[j]$
- 递推公式： $dp[j] = \max(dp[j], dp[j - weight[i]] + value[i])$, 即不放物品*i*和放物品*i*
- 初始化：全0即可
- 遍历顺序：先物品后重量（重量倒序遍历）

倒序遍历是为了保证物品i只被放入一次！但如果一旦正序遍历了，那么物品0就会被重复加入多次！

举一个例子：物品0的重量 $weight[0] = 1$ ，价值 $value[0] = 15$

如果正序遍历

$$dp[1] = dp[1 - weight[0]] + value[0] = 15$$

$$dp[2] = dp[2 - weight[0]] + value[0] = 30$$

此时 $dp[2]$ 就已经是30了，意味着物品0，被放入了两次，所以不能正序遍历。

为什么倒序遍历，就可以保证物品只放入一次呢？

倒序就是先算 $dp[2]$

$$dp[2] = dp[2 - weight[0]] + value[0] = 15 \quad (dp\text{数组已经都初始化为0})$$

$$dp[1] = dp[1 - weight[0]] + value[0] = 15$$

所以从后往前循环，每次取得状态不会和之前取得状态重合，这样每种物品就只取一次了。

倒序遍历的原因是，本质上还是一个对二维数组的遍历，并且右下角的值依赖上一层左上角的值，因此需要保证左边的值仍然是上一层的，从右向左覆盖。

```
void test_1_wei_bag_problem() {
    vector<int> weight = {1, 3, 4};
    vector<int> value = {15, 20, 30};
    int bagWeight = 4;

    // 初始化
    vector<int> dp(bagWeight + 1, 0);
    for(int i = 0; i < weight.size(); i++) { // 遍历物品
        for(int j = bagWeight; j >= weight[i]; j--) { // 遍历背包容量
            dp[j] = max(dp[j], dp[j - weight[i]] + value[i]);
        }
    }
    cout << dp[bagWeight] << endl;
}

int main() {
    test_1_wei_bag_problem();
}
```

416. 分割等和子集

难度 中等 击 1755     

给你一个 只包含正整数 的 非空 数组 `nums` 。请你判断是否可以将这个数组分割成两个子集，使得两个子集的元素和相等。

示例 1：

输入: `nums = [1,5,11,5]`

输出: `true`

解释: 数组可以分割成 `[1, 5, 5]` 和 `[11]` 。

示例 2：

输入: `nums = [1,2,3,5]`

输出: `false`

解释: 数组不能分割成两个元素和相等的子集。

提示:

- `1 <= nums.length <= 200`
- `1 <= nums[i] <= 100`

```
① C++ • 智能模式 ② 模拟面试 | i P ⌂ >_ ⌂ ⌂
```

```
1 class Solution {
2 public:
3     bool canPartition(vector<int>& nums) {
4         int sums = accumulate(nums.begin(), nums.end(), 0);
5         if (sums % 2 == 1) return false;
6         vector<int> dp(sums/2 + 1, 0);
7         for (int i = 0; i < nums.size(); i++) {
8             for (int j = sums/2; j >= nums[i]; j--) {
9                 dp[j] = max(dp[j], dp[j-nums[i]] + nums[i]);
10            }
11        }
12        return dp[sums/2] == sums/2;
13    }
14 }
```

递推公式决定了 $dp[i]$ 不可能大于 i

因为 $dp[i]$ 不可能大于 i , 所以遍历完后, 直接进行对比 $dp[sums/2] == sums/2$, 如果相等, 说明刚好装满, 有可以平分数组和的子集

1. $dp[j]$ 含义: 容量为 j 的背包, 所背的物品价值最大可以为 $dp[j]$
2. 递推公式: $dp[j] = \max(dp[j], dp[j-nums[i]] + nums[i])$
3. 初始化: 全0即可
4. 遍历顺序: 外层遍历 $nums$ 的数字, 顺序, 内层遍历背包重量, 倒序

.8 买卖股票的最佳时机 (121、122、123、188)

- 121

121. 买卖股票的最佳时机

难度 简单 收藏 2969 分享

给定一个数组 `prices`，它的第 `i` 个元素 `prices[i]` 表示一支给定股票第 `i` 天的价格。

你只能选择 某一天 买入这只股票，并选择在 未来的某一个不同的日子 卖出该股票。设计一个算法来计算你所能获取的最大利润。

返回你可以从这笔交易中获取的最大利润。如果你不能获取任何利润，返回 `0`。

示例 1：

输入: `[7,1,5,3,6,4]`

输出: 5

解释: 在第 2 天 (股票价格 = 1) 的时候买入，在第 5 天 (股票价格 = 6) 的时候卖出，最大利润 = $6 - 1 = 5$ 。

注意利润不能是 $7 - 1 = 6$ ，因为卖出价格需要大于买入价格；同时，你不能在买入前卖出股票。

示例 2：

输入: `prices = [7,6,4,3,1]`

输出: 0

解释: 在这种情况下，没有交易完成，所以最大利润为 0。

提示:

- $1 \leq \text{prices.length} \leq 10^5$
- $0 \leq \text{prices}[i] \leq 10^4$

```
1 class Solution {
2     public:                                            贪心解法：取左最小，取右最大
3         int maxProfit(vector<int>& prices) {
4             int low = INT_MAX;
5             int result = 0;
6             for (int i = 0; i < prices.size(); i++) {
7                 low = min(low, prices[i]); // 取最左最小价格
8                 result = max(result, prices[i] - low); // 直接取最大区间利润
9             }
10            return result;
11        }
12    };
```

```
① C++ • 智能模式 ② 模拟面试 | i P ⌂ ⌄ ⌅ ⌆ ⌇
```

```
1 class Solution {
2 public:
3     int maxProfit(vector<int>& prices) {
4         int days = prices.size();
5         vector<vector<int>> dp(days, vector<int> (2, 0));
6         dp[0][0] = -prices[0];
7         for (int i = 1; i < days; i++) {
8             dp[i][0] = max(dp[i-1][0], -prices[i]);
9             dp[i][1] = max(dp[i-1][1], dp[i-1][0] + prices[i]);
10        }
11    return dp[days-1][1];
12 }
13 };
```

1. dp含义：`dp[i][0]`表示第*i*天持有股票所得最多现金，`dp[i][0]`是非正数，相当于去找买入的最低价，要么第*i*-1天也持有，要么第*i*天买入，看哪个更大；`dp[i][1]`表示第*i*天不持有股票所得最多现金，要么第*i*-1天也不持有，要么第*i*天卖出赚钱，看哪个更大
2. 递推公式：如上图所示
3. 初始化：`dp[i][0]`和`dp[i][1]`的推导都是根据`dp[i-1]`，所以只需初始化`dp[0]`，`dp[0][0]`是第1天就买入，`dp[0][1]`是第1天不买入
4. 顺序：从前往后

122. 买卖股票的最佳时机 II

难度 中等 山 2122 ☆ ⚡ 文 🔔 📄

给你一个整数数组 `prices`，其中 `prices[i]` 表示某支股票第 `i` 天的价格。

在每一天，你可以决定是否购买和/或出售股票。你在任何时候 **最多** 只能持有 **一股** 股票。你也可以先购买，然后在同一天出售。

返回 你能获得的 **最大** 利润。

示例 1：

输入: `prices = [7,1,5,3,6,4]`

输出: 7

解释: 在第 2 天 (股票价格 = 1) 的时候买入，在第 3 天 (股票价格 = 5) 的时候卖出，这笔交易所能获得利润 = $5 - 1 = 4$ 。

随后，在第 4 天 (股票价格 = 3) 的时候买入，在第 5 天 (股票价格 = 6) 的时候卖出，这笔交易所能获得利润 = $6 - 3 = 3$ 。

总利润为 $4 + 3 = 7$ 。

示例 2：

输入: `prices = [1,2,3,4,5]`

输出: 4

解释: 在第 1 天 (股票价格 = 1) 的时候买入，在第 5 天 (股票价格 = 5) 的时候卖出，这笔交易所能获得利润 = $5 - 1 = 4$ 。

总利润为 4。

示例 3：

输入: `prices = [7,6,4,3,1]`

输出: 0

解释: 在这种情况下，交易无法获得正利润，所以不参与交易可以获得最大利润，最大利润为 0。

提示：

- $1 \leq \text{prices.length} \leq 3 * 10^4$
- $0 \leq \text{prices}[i] \leq 10^4$

• 智能模式

模拟面试 | i P C >_ ☰ []

```
1 class Solution {
2 public:
3     int maxProfit(vector<int>& prices) {
4         int days = prices.size();
5         vector<vector<int>> dp(days, vector<int> (2, 0));
6         dp[0][0] = -prices[0];
7         for (int i = 1; i < days; i++) {
8             dp[i][0] = max(dp[i-1][0], dp[i-1][1] - prices[i]);
9             dp[i][1] = max(dp[i-1][1], dp[i-1][0] + prices[i]);
10        }
11        return dp[days-1][1];
12    }
13};
```

1. dp含义：同上题
2. 递推公式：同上题，仅 $dp[i][0]$ 改变为 $dp[i][0] = \max(dp[i-1][0], dp[i-1][1] - prices[i])$ ，即可以买卖多次了，第*i*天持有股票的最大现金收益可能等于*i*-1天不持有股票减第*i*天买入股票，引入了之前交易的利润
3. 初始化：同上题
4. 遍历顺序：同上题

• 123

123. 买卖股票的最佳时机 III

难度 困难 1408 ☆ ⌂ 文 ⌂

给定一个数组，它的第 i 个元素是一支给定的股票在第 i 天的价格。

设计一个算法来计算你所能获取的最大利润。你最多可以完成 两笔 交易。

注意：你不能同时参与多笔交易（你必须在再次购买前出售掉之前的股票）。

示例 1：

输入: `prices = [3,3,5,0,0,3,1,4]`

输出: 6

解释: 在第 4 天 (股票价格 = 0) 的时候买入，在第 6 天 (股票价格 = 3) 的时候卖出，这笔交易所能获得利润 = $3-0 = 3$ 。

随后，在第 7 天 (股票价格 = 1) 的时候买入，在第 8 天 (股票价格 = 4) 的时候卖出，这笔交易所能获得利润 = $4-1 = 3$ 。

示例 2：

输入: `prices = [1,2,3,4,5]`

输出: 4

解释: 在第 1 天 (股票价格 = 1) 的时候买入，在第 5 天 (股票价格 = 5) 的时候卖出，这笔交易所能获得利润 = $5-1 = 4$ 。

注意你不能在第 1 天和第 2 天接连购买股票，之后再将它们卖出。

因为这样属于同时参与了多笔交易，你必须在再次购买前出售掉之前的股票。

示例 3：

输入: `prices = [7,6,4,3,1]`

输出: 0

解释: 在这个情况下，没有交易完成，所以最大利润为 0。

示例 4：

输入: `prices = [1]`

输出: 0

提示：

- $1 \leq \text{prices.length} \leq 10^5$
- $0 \leq \text{prices}[i] \leq 10^5$

```
① C++ • 智能模式 ② 模拟面试 | i P ⌂ >_ ⌂ ⌂
```

```
1 class Solution {
2 public:
3     int maxProfit(vector<int>& prices) {
4         int days = prices.size();
5         vector<vector<int>> dp(days, vector<int> (5, 0));
6         dp[0][1] = -prices[0];
7         dp[0][3] = -prices[0];
8         for (int i = 1; i < days; i++) {
9             dp[i][1] = max(dp[i-1][1], 0 - prices[i]);
10            dp[i][2] = max(dp[i-1][2], dp[i-1][1] + prices[i]);
11            dp[i][3] = max(dp[i-1][3], dp[i-1][2] - prices[i]);
12            dp[i][4] = max(dp[i-1][4], dp[i-1][3] + prices[i]);
13        }
14        return dp[days-1][4];
15    }
16};
```

1. dp含义：`dp[i][1]`第*i*天第一次持有股票的最大现金收益，`dp[i][2]`第*i*天第一次不持有股票的最大现金收益，`dp[i][3]`第*i*天第二次持有股票的最大现金收益，`dp[i][4]`第*i*天第二次不持有股票的最大现金收益。
2. 递推公式：如上图所示
3. 初始化：`dp[0][1]`第一天第一次持有股票，初始化为`-prices[0]`，`dp[0][3]`第一天第二次持有股票，初始化为`-prices[0]`，可以理解为第一天买入卖出再买入的情况（本题一定会进行第二次买入卖出，因为可以当天买入卖出赚0元来凑次数，所以答案也是取`dp[days-1][4]`）
4. 遍历顺序：从前往后