

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 10 月 1 日 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++ 模拟面试 | i P ⌂ >_ ⚙ []  
• 智能模式  
  
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++) {
15                result.push_back(nums[i]*nums[i]); // 全非负情况
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--) {
22                result.push_back(nums[i]*nums[i]); // 全负情况 / 全非正情况
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]);
30                q++;
31            } else {           // 从中间向两边扩散，小的放入result
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    };
```

977解法2——双指针法，从两侧往中间

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)

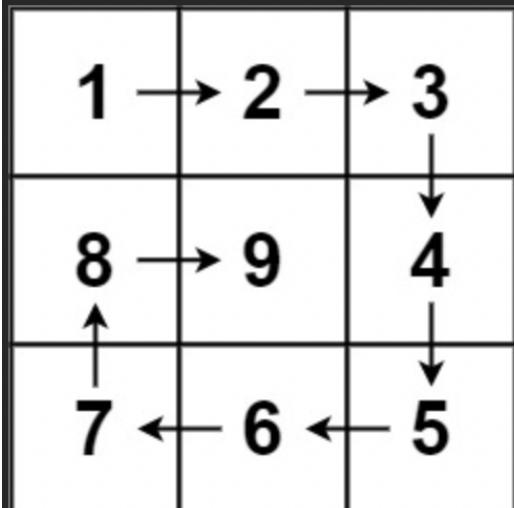
- 59

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};
```

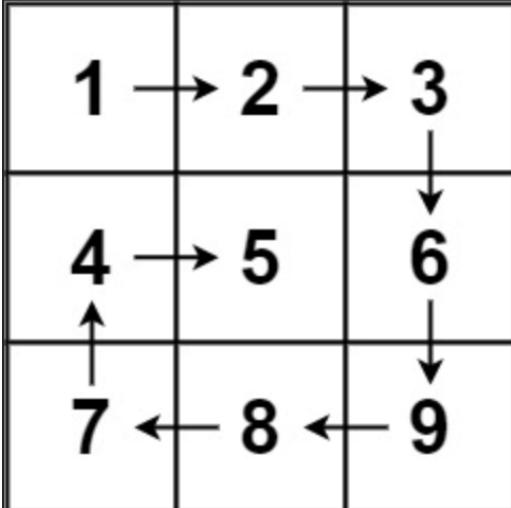
• 54

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：

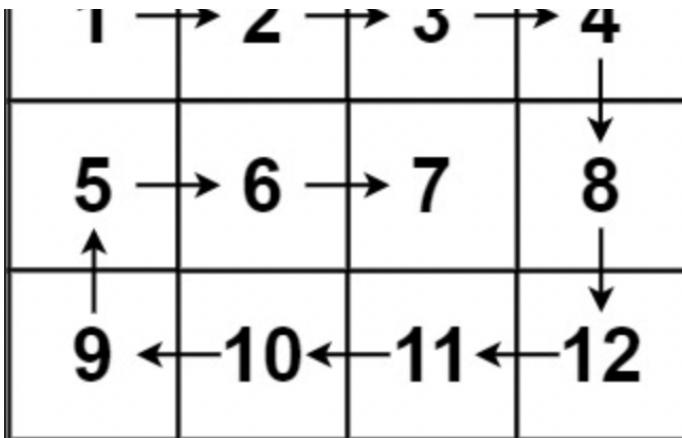


输入: $\text{matrix} = [[1,2,3],[4,5,6],[7,8,9]]$

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）

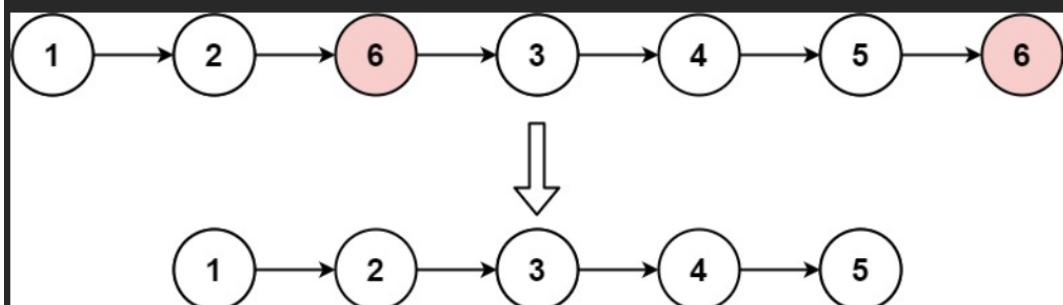
- 203

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）

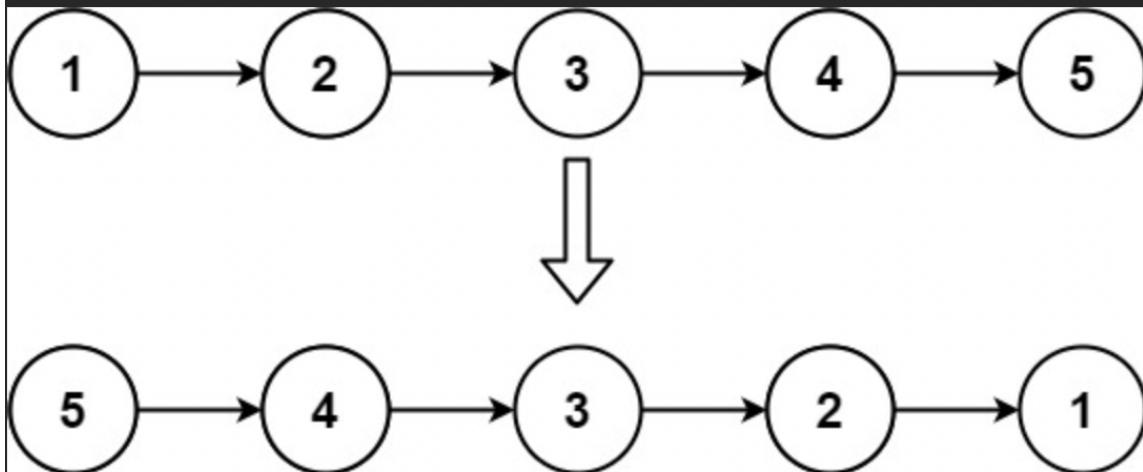
- 206

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。

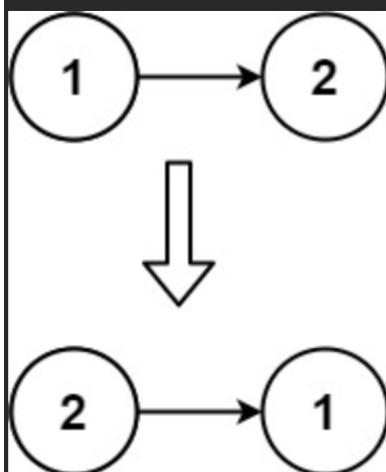
示例 1：



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

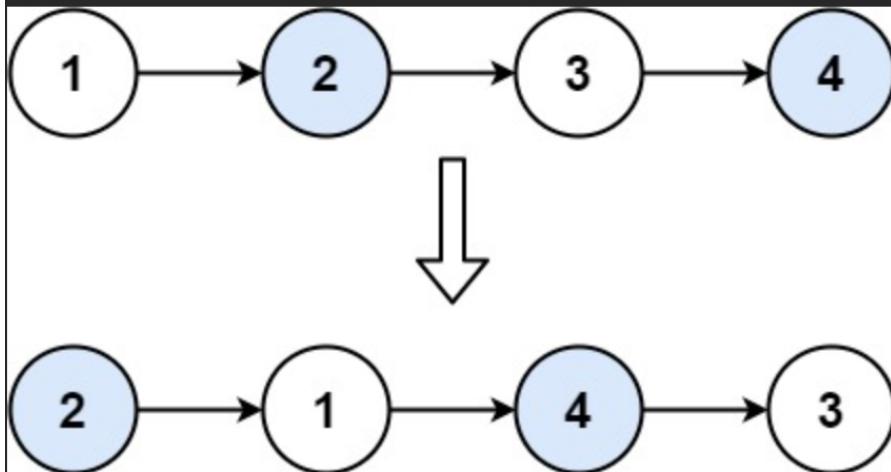
切换为英文

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 文本

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 白 1489 ★ ⚖ 文 A 🔍

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：

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

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

```
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) {
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 };
```

计数法解决
定义一个string计数每个字符串

```
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;      此方法简单但时间复杂度高
        for (auto it = mp.begin(); it != mp.end(); ++it) {
            ans.emplace_back(it->second);
        }
        return ans;
    }
};
```

排序法

- 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[j]];  
16             }  
17         }  
18     }  
19 }
```

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 ★ ⌂ ⚡A 🔔 ⌂

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 $i \neq j$ 、 $i \neq k$ 且 $j \neq k$ ，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：

- $3 \leq \text{nums.length} \leq 3000$
- $-10^5 \leq \text{nums}[i] \leq 10^5$

```
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;
}

```

- 611 (番外篇，求三角形个数)

611. 有效三角形的个数

已解答

中等 相关标签 相关企业 A*

给定一个包含非负整数的数组 `nums`，返回其中可以组成三角形三条边的三元组个数。

示例 1:

输入: `nums = [2,2,3,4]`

输出: 3

解释: 有效的组合是：

`2,3,4` (使用第一个 2)

`2,3,4` (使用第二个 2)

`2,2,3`

示例 2:

输入: `nums = [4,2,3,4]`

输出: 4

提示:

• `1 <= nums.length <= 1000`

• `0 <= nums[i] <= 1000`

</> 代码

Python3 智能模式

三 取 ⌂ ↗

```
1 class Solution:
2     def triangleNumber(self, nums: List[int]) -> int:
3         length = len(nums)
4         if length <= 2:
5             return 0
6         new_nums = sorted(nums) 先sort
7         result = 0
8         for i in range(0, length - 2, 1):
9             for j in range(i + 1, length - 1, 1): 固定两个数
10                left, right, k = j + 1, length - 1, j
11                while left <= right:
12                    mid = left + (right - left) // 2
13                    if new_nums[i] + new_nums[j] > new_nums[mid]:
14                        k = mid
15                        left = mid + 1
16                    else:
17                        right = mid - 1
18                    result += (k - j)
19
20 return result
```

二分查找，找到第三个数
条件是 $\text{nums}[i] + \text{nums}[j]$ 大于
 $\text{nums}[\text{mid}]$ 的最大 mid

不二分查找超时！！

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++;
                }
            }
        }
        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    }  
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

6、动态规划 (dp)

6.1 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



公众号：代码随想录

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

6.2 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 C++ • 智能模式

② 模拟面试 | i P ⌂ > ⌂ ⌂

```
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     }
}
```

6.3 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 }
```

6.4 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};
```

6.5 不同路径 (62、63)

• 62

62. 不同路径

难度 中等 1763 文

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右移一步。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

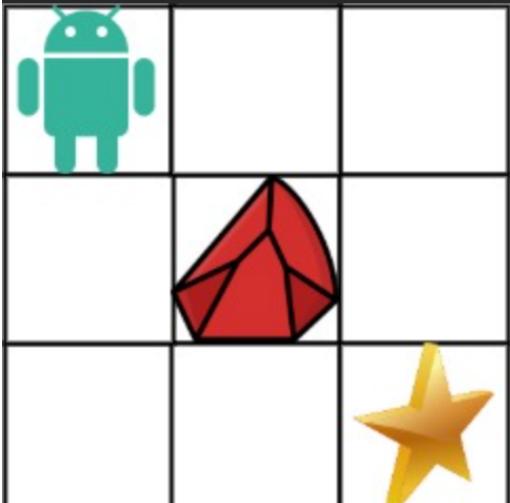
一个机器人位于一个 $m \times n$ 网格的左上角（起始点在下图中标记为“Start”）。

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]]`

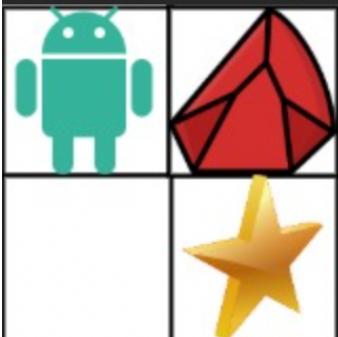
输出: 2

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 }
```

6.6 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:

- $2 \leq n \leq 58$

● 智能模式

④ 模拟面试 | i ▶ ⌂ ⌂ ⌂ ⌂ ⌂ ⌂

```

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           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       } 时考虑到了
14   };
15 }
```

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

补充：dp[6]的24拆分，为什么不dp[2]dp[4]，因为dp[2]会拆成11，已经在1dp[5]考虑过了；同理dp[3]dp[3]，前面的dp[3]已经在2dp[4]考虑过了，因为3会拆成12和21，就算拆成111也在1dp[5]考虑过了

6.7 0-1背包问题

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： $dp[i][0] = 0$ ，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_problem1() {
    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 (01背包是否能装满，装满True)

416. 分割等和子集

难度 中等 1755 文

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++ ② 智能模式 ③ 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

- 1049 (dp含义同上, 本题其实就是要尽量让石头分成重量相同的两堆, 相撞之后剩下的石头最小, 这样就化成01背包问题了)

1049. 最后一块石头的重量 II

中等 相关标签 相关企业 提示 A_文

有一堆石头，用整数数组 `stones` 表示。其中 `stones[i]` 表示第 `i` 块石头的重量。

每一回合，从中选出任意两块石头，然后将它们一起粉碎。假设石头的重量分别为 `x` 和 `y`，且 `x <= y`。那么粉碎的可能结果如下：

- 如果 `x == y`，那么两块石头都会被完全粉碎；
- 如果 `x != y`，那么重量为 `x` 的石头将会完全粉碎，而重量为 `y` 的石头新重量为 `y-x`。

最后，最多只会剩下一块石头。返回此石头 **最小的可能重量**。如果没有石头剩下，就返回 `0`。

示例 1：

输入: `stones = [2,7,4,1,8,1]`

输出: 1

解释:

组合 2 和 4，得到 2，所以数组转化为 `[2,7,1,8,1]`，

组合 7 和 8，得到 1，所以数组转化为 `[2,1,1,1]`，

组合 2 和 1，得到 1，所以数组转化为 `[1,1,1]`，

组合 1 和 1，得到 0，所以数组转化为 `[1]`，这就是最优值。

示例 2：

输入: `stones = [31,26,33,21,40]`

输出: 5

提示:

- `1 <= stones.length <= 30`
- `1 <= stones[i] <= 100`

</> 代码

Python3 ▾ 智能模式

三 口 5

```
1 class Solution:
2     def lastStoneWeightII(self, stones: List[int]) -> int:
3         sums = sum(stones)
4         target = sums // 2
5         dp = [0] * (target + 1)
6         for i in range(len(stones)):
7             for j in range(target, stones[i] - 1, -1):
8                 dp[j] = max(dp[j], dp[j - stones[i]] + stones[i])
9         return sums - 2 * dp[target]
```

- 494 (01背包有多少种方式装满)

494. 目标和

已解答

中等 相关标签 相关企业 A_x

给你一个非负整数数组 `nums` 和一个整数 `target`。

向数组中的每个整数前添加 `'+'` 或 `'-'`，然后串联起所有整数，可以构造一个 表达式：

- 例如，`nums = [2, 1]`，可以在 `2` 之前添加 `'+'`，在 `1` 之前添加 `'-'`，然后串联起来得到表达式 `"+2-1"`。

返回可以通过上述方法构造的、运算结果等于 `target` 的不同 表达式 的数目。

示例 1：

输入: `nums = [1,1,1,1,1]`, `target = 3`

输出: 5

解释: 一共有 5 种方法让最终目标和为 3。

`-1 + 1 + 1 + 1 + 1 = 3`

`+1 - 1 + 1 + 1 + 1 = 3`

`+1 + 1 - 1 + 1 + 1 = 3`

`+1 + 1 + 1 - 1 + 1 = 3`

`+1 + 1 + 1 + 1 - 1 = 3`

示例 2：

输入: `nums = [1]`, `target = 1`

输出: 1

提示:

- `1 <= nums.length <= 20`
- `0 <= nums[i] <= 1000`
- `0 <= sum(nums[i]) <= 1000`
- `-1000 <= target <= 1000`

</> 代码

Python3 ▾ 🔒 智能模式

☰ ⌂ ⌂

```
1 class Solution:
2     def findTargetSumWays(self, nums: List[int], target: int) -> int:
3         """
4             dp[j] 表示用若干元素装满容量为 j 的背包，装法有 dp[j] 种
5             递推公式: dp[j] = dp[j] + dp[j - nums[i]]，理解递推公式，对于当前的容量 j 和目前遍历
6             到的数字 nums[i]，如果不装 nums[i]，那么方法数就等于 dp[j]，原来咋装的就咋装，方法不变，如果装上
7             nums[i]，则方法数就等于 dp[j - nums[i]]，所以 dp[j] 是两者相加，又由于 dp[j] 依赖于 dp[j - nums
8             [i]]，所以内层必须逆序遍历
9             初始化: dp[0] = 1，因为当集合 {0} 且 target=0 时，dp[0] = 1
10            ...
11            sums = sum(nums)
12            if (sums + target) % 2 == 1 or (sums + target) < 0:
13                return 0
14            left = (sums + target) // 2
15            dp = [0] * (left + 1)
16            dp[0] = 1
17            for i in range(len(nums)):
18                for j in range(left, nums[i] - 1, -1):
19                    dp[j] += dp[j - nums[i]]
```

初始化 dp[0] 看卡尔视频

递推公式看卡尔视频评论1

假设所有取正数的为 left, 所有取负数的为 right, left+right=sums, left-right=target, 所以 left=(sums+target)/2

- 474 (01 背包二维背包容量, 最多能装多少个物品)

474. 一和零

已解答 

中等 相关标签 相关企业 A₂

给你一个二进制字符串数组 `strs` 和两个整数 `m` 和 `n`。

请你找出并返回 `strs` 的最大子集的长度，该子集中最多有 `m` 个 `0` 和 `n` 个 `1`。

如果 `x` 的所有元素也是 `y` 的元素，集合 `x` 是集合 `y` 的 子集。

示例 1：

输入: `strs = ["10", "0001", "111001", "1", "0"]`, `m = 5`, `n = 3`

输出: 4

解释: 最多有 5 个 0 和 3 个 1 的最大子集是 `{"10", "0001", "1", "0"}`，因此答案是 4。
其他满足题意但较小的子集包括 `{"0001", "1"}` 和 `{"10", "1", "0"}`。`{"111001"}` 不满足题意，因为它含 4 个 1，大于 n 的值 3。

示例 2：

输入: `strs = ["10", "0", "1"]`, `m = 1`, `n = 1`

输出: 2

解释: 最大的子集是 `{"0", "1"}`，所以答案是 2。

提示：

- `1 <= strs.length <= 600`
- `1 <= strs[i].length <= 100`
- `strs[i]` 仅由 `'0'` 和 `'1'` 组成
- `1 <= m, n <= 100`

```

1 class Solution:
2     def findMaxForm(self, strs: List[str], m: int, n: int) -> int:
3         """
4             dp[i][j] 代表最多包含 i 和 j 个 1 的最大子集大小
5             递推公式: dp[i][j] = max(dp[i][j], dp[i-zeroes][j-ones]+1), 对每个 str, 如果加上
6             其 01 数量直接不满足 mn 条件, 则不会进入循环, 如果满足, 则进入循环, 计算不算和算上哪个数量大
7             初始化: 全是 0, 可以把 mn 看作一个二维背包容量, 每个 str 的 01 数量作为每个物体的重量, 每个
8             str 作为一个物体
9             ...
10            dp = [[0] * (n+1) for _ in range(m+1)]
11            for each in strs:
12                zeroes = each.count('0')
13                ones = each.count('1')
14                for i in range(m, zeroes-1, -1):
15                    for j in range(n, ones-1, -1):
16                        dp[i][j] = max(dp[i][j], dp[i-zeroes][j-ones]+1)
17
18            return dp[m][n]

```

6.8 完全背包问题

完全背包问题题目描述: 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 $value[i]$ 。每件物品可以用无限次, 求解将哪些物品装入背包里物品价值总和最大。**无限次的条件与 01 背包只差在遍历顺序上 ! !**

```

# 一维dp的内层循环正序遍历, 表示一个物品可以拿多次, 且顺序可颠倒, 原因看代码随想录笔记
for i in range(len(weight)): # 遍历物品
    for j in range(weight[i], bagWeight + 1): # 遍历背包容量
        dp[j] = max(dp[j], dp[j - weight[i]] + value[i])

```

- 52 ([卡码网KamaCoder](#))

52. 携带研究材料（第七期模拟笔试）

时间限制: 1.000S 空间限制: 128MB

题目描述

小明是一位科学家，他需要参加一场重要的国际科学大会，以展示自己的最新研究成果。他需要带一些研究材料，但是他的行李箱空间有限。这些研究材料包括实验设备、文献资料和实验样本等等，它们各自占据不同的重量，并且具有不同的价值。

小明的行李箱所能承担的总重量为 N ，问小明应该如何抉择，才能携带最大价值的研究材料，每种研究材料可以选择无数次，并且可以重复选择。

输入描述

第一行包含两个整数， N , V ，分别表示研究材料的种类和行李空间

接下来包含 N 行，每行两个整数 w_i 和 v_i ，代表第 i 种研究材料的重量和价值

输出描述

输出一个整数，表示最大价值。

输入示例

```
4 5
1 2
2 4
3 4
4 5
```

输出示例

```
10
```

提示信息

第一种材料选择五次，可以达到最大值。

数据范围：

$1 \leq N \leq 10000$;
 $1 \leq V \leq 10000$;
 $1 \leq w_i, v_i \leq 10^9$.

Python



运行

提交



```
1 N, V = map(int, input().split())
2 w, v = [0] * N, [0] * N
3 for i in range(N):
4     w[i], v[i] = map(int, input().split())
5 dp = [0] * (V+1)
6 for i in range(N):
7     for j in range(w[i], V+1):
8         dp[j] = max(dp[j], dp[j-w[i]]+v[i])
9
10 print(dp[V])
11
12 |
```

自定义运行数据

自定义输入:

4 5
1 2
2 4
3 4
4 5

运行结果:

10

- 518 (组合)

518. 零钱兑换 II

已解答

中等

相关标签

相关企业

A₂

给你一个整数数组 `coins` 表示不同面额的硬币，另给一个整数 `amount` 表示总金额。

请你计算并返回可以凑成总金额的硬币组合数。如果任何硬币组合都无法凑出总金额，返回 `0`。

假设每一种面额的硬币有无限个。

题目数据保证结果符合 32 位带符号整数。

示例 1：

输入: `amount = 5, coins = [1, 2, 5]`

输出: 4

解释: 有四种方式可以凑成总金额:

$5=5$

$5=2+2+1$

$5=2+1+1+1$

$5=1+1+1+1+1$

示例 2：

输入: `amount = 3, coins = [2]`

输出: 0

解释: 只用面额 2 的硬币不能凑成总金额 3。

示例 3：

输入: `amount = 10, coins = [10]`

输出: 1

提示:

- `1 <= coins.length <= 300`
- `1 <= coins[i] <= 5000`
- `coins` 中的所有值互不相同
- `0 <= amount <= 5000`

</> 代码

Python3 ✓ 🔒 智能模式

☰ ⌂ ⌂

```
1 class Solution:
2     def change(self, amount: int, coins: List[int]) -> int:
3         dp = [0] * (amount + 1)
4         dp[0] = 1
5         for i in range(len(coins)):
6             for j in range(coins[i], amount + 1):
7                 dp[j] += dp[j - coins[i]] 考虑组合数量的遍历公式统一写法
8         return dp[amount]
```

至于i和j的遍历顺序，如果先物品后重量，添加完coins[0]后才添加coins[1]，所以只会出现{coins[0], coins[1]}的情况，求的是组合（无序）

已存储至本地 如果先重量后物品，在重量k时，会考虑所有物品，假设此时考虑了coins[1]，在重量k+1时，又会从coins[0]开始遍历，会出现{coins[1], coins[0]}的情况，求的是排列（有序）
行 7, 列 42

- 377 (排列)

377. 组合总和 IV

已解答

中等

相关标签

相关企业

A_x

给你一个由 **不同** 整数组成的数组 `nums`，和一个目标整数 `target`。请你从 `nums` 中找出并返回总和为 `target` 的元素组合的个数。

题目数据保证答案符合 32 位整数范围。

示例 1：

输入: `nums = [1,2,3]`, `target = 4`

输出: 7

解释:

所有可能的组合为:

(1, 1, 1, 1)

(1, 1, 2)

(1, 2, 1)

(1, 3)

(2, 1, 1)

(2, 2)

(3, 1)

请注意，顺序不同的序列被视作不同的组合。

示例 2：

输入: `nums = [9]`, `target = 3`

输出: 0

提示:

- `1 <= nums.length <= 200`
- `1 <= nums[i] <= 1000`
- `nums` 中的所有元素 **互不相同**
- `1 <= target <= 1000`

</> 代码

Python3 智能模式

三 口 ⌂

```
1 class Solution:
2     def combinationSum4(self, nums: List[int], target: int) -> int:
3         dp = [0] * (target + 1)
4         dp[0] = 1
5         for j in range(target + 1):
6             for i in nums:          排列：先重量后物品
7                 if (j - i) >= 0:
8                     dp[j] += dp[j - i]
9         return dp[target]
```

已存储至本地

行 8, 列 21

- 57 ([卡码网KamaCoder](#))

57. 爬楼梯 (第八期模拟笔试)

时间限制: 1.000S 空间限制: 128MB

题目描述

假设你正在爬楼梯。需要 n 阶你才能到达楼顶。
每次你可以爬至多 m ($1 \leq m < n$) 个台阶。你有多少种不同的方法可以爬到楼顶呢?
注意：给定 n 是一个正整数。

输入描述

输入共一行，包含两个正整数，分别表示 n, m

输出描述

输出一个整数，表示爬到楼顶的方法数。

输入示例

3 2

输出示例

3

提示信息

数据范围：

$1 \leq m < n \leq 32$;

当 $m = 2, n = 3$ 时， $n = 3$ 这表示一共有三个台阶， $m = 2$ 代表你每次可以爬一个台阶或者两个台阶。

此时你有三种方法可以爬到楼顶。

1. 1 阶 + 1 阶 + 1 阶段

2. 1 阶 + 2 阶

3. 2 阶 + 1 阶

Python



运行

提交



```
1 n, m = map(int, input().split())
2 dp = [0] * (n + 1)
3 dp[0] = 1
4 for j in range(n+1):
5     for i in range(1, m + 1):
6         if j >= i:
7             dp[j] += dp[j - i]
8 print(dp[n])
```

322. 零钱兑换

已解答

中等

相关标签

相关企业

A+

给你一个整数数组 `coins`，表示不同面额的硬币；以及一个整数 `amount`，表示总金额。

计算并返回可以凑成总金额所需的 最少的硬币个数。如果没有任何一种硬币组合能组成总金额，返回 `-1`。

你可以认为每种硬币的数量是无限的。

示例 1：

输入: `coins = [1, 2, 5]`, `amount = 11`

输出: `3`

解释: $11 = 5 + 5 + 1$

示例 2：

输入: `coins = [2]`, `amount = 3`

输出: `-1`

示例 3：

输入: `coins = [1]`, `amount = 0`

输出: `0`

提示：

- $1 \leq \text{coins.length} \leq 12$
- $1 \leq \text{coins}[i] \leq 2^{31} - 1$
- $0 \leq \text{amount} \leq 10^4$

`</>` 代码

Python3 ▾ 🔒 智能模式



```

1 class Solution:
2     def coinChange(self, coins: List[int], amount: int) -> int:
3         dp = [float('inf')] * (amount + 1)  初始化为最大值
4         dp[0] = 0
5         for each in coins:    遍历顺序无所谓，因为是求最小值，所以排列 / 组合不影响
6             for j in range(each, amount + 1):
7                 if dp[j - each] != float('inf'):
8                     dp[j] = min(dp[j - each] + 1, dp[j])
9         return dp[amount] if dp[amount] != float('inf') else -1

```

279. 完全平方数

已解答

中等 相关标签 相关企业 A_x

给你一个整数 n ，返回 和为 n 的完全平方数的最少数量。

完全平方数 是一个整数，其值等于另一个整数的平方；换句话说，其值等于一个整数自乘的积。例如，1、4、9 和 16 都是完全平方数，而 3 和 11 不是。

示例 1：

输入: $n = 12$

输出: 3

解释: $12 = 4 + 4 + 4$

示例 2：

输入: $n = 13$

输出: 2

解释: $13 = 4 + 9$

提示:

- $1 \leq n \leq 10^4$

代码

Python3 智能模式

三 口 5

```
1 class Solution:
2     def numSquares(self, n: int) -> int:
3         import math
4         dp = [n] * (n + 1)
5         sq = [i**2 for i in range(1, int(math.sqrt(n)) + 1)]
6
7         for each in sq:          此为自己思路，官网思路大致相同，求最小值
8             dp[each] = 1          与遍历顺序无关，与上题一样
9
10        for each in sq:
11            for j in range(each, n+1):
12                dp[j] = min(dp[j], dp[j-each] + dp[each])
13
14        return dp[n]
```

139. 单词拆分

已解答

中等 相关标签 相关企业

给你一个字符串 `s` 和一个字符串列表 `wordDict` 作为字典。如果可以利用字典中出现的一个或多个单词拼接出 `s` 则返回 `true`。

注意：不要求字典中出现的单词全部都使用，并且字典中的单词可以重复使用。

示例 1：

输入: `s = "leetcode"`, `wordDict = ["leet", "code"]`

输出: `true`

解释: 返回 `true` 因为 `"leetcode"` 可以由 `"leet"` 和 `"code"` 拼接成。

示例 2：

输入: `s = "applepenapple"`, `wordDict = ["apple", "pen"]`

输出: `true`

解释: 返回 `true` 因为 `"applepenapple"` 可以由 `"apple"` "pen" "apple" 拼接成。
注意，你可以重复使用字典中的单词。

示例 3：

输入: `s = "catsandog"`, `wordDict = ["cats", "dog", "sand", "and", "cat"]`

输出: `false`

提示：

- `1 <= s.length <= 300`
- `1 <= wordDict.length <= 1000`
- `1 <= wordDict[i].length <= 20`
- `s` 和 `wordDict[i]` 仅由小写英文字母组成
- `wordDict` 中的所有字符串 互不相同

</> 代码

Python3 ✓ 智能模式

↶ ⌂ ⌂ ⌂

```
1 class Solution:
2     def wordBreak(self, s: str, wordDict: List[str]) -> bool:
3         dp = [0] * (len(s) + 1)
4         dp[0] = 1
5         for i in range(1, len(s) + 1):
6             for j in range(i):
7                 if dp[j] and s[j:i] in wordDict:
8                     dp[i] = 1
9         return bool(dp[len(s)])
```

其实不把这个问题当作完全背包，代码也是这么写

$dp[i]$: 包涵当前位置的向前长度为*i*的子串是不是能被wordDict组成

初始化: $dp[0]$ 为1, 其余为0

递推公式: 当前 $dp[i]$ 与 $dp[j]$ 和 $s[j:i]$ 有关, 如果 $dp[j]=1$ 且 $s[j:i]$ 还在wordDict中, 则 $dp[i]=1$

已存储至本地

行 7, 列 49

6.9 多重背包问题 (基本不会考, 只一个例题, 解法同01)

- 56 ([卡码网KamaCoder](#))

56. 携带矿石资源（第八期模拟笔试）

时间限制: 5.000S 空间限制: 256MB

题目描述

你是一名宇航员，即将前往一个遥远的行星。在这个行星上，有许多不同类型的矿石资源，每种矿石都有不同的重要性和价值。你需要选择哪些矿石带回地球，但你的宇航舱有一定的容量限制。给定一个宇航舱，最大容量为 C 。现在有 N 种不同类型的矿石，每种矿石有一个重量 $w[i]$ ，一个价值 $v[i]$ ，以及最多 $k[i]$ 个可用。不同类型的矿石在地球上的市场价值不同。你需要计算如何在不超过宇航舱容量的情况下，最大化你所能获取的总价值。

输入描述

输入共包括四行，第一行包含两个整数 C 和 N ，分别表示宇航舱的容量和矿石的种类数量。
接下来的三行，每行包含 N 个正整数。具体如下：
第二行包含 N 个整数，表示 N 种矿石的重量。
第三行包含 N 个整数，表示 N 种矿石的价格。
第四行包含 N 个整数，表示 N 种矿石的可用数量上限。

输出描述

输出一个整数，代表获取的最大价值。

输入示例

```
10 3
1 3 4
15 20 30
2 3 2
```

输出示例

```
90
```

提示信息

数据范围：
 $1 \leq C \leq 10000$;
 $1 \leq N \leq 10000$;
 $1 \leq w[i], v[i], k[i] \leq 10000$;

Python



运行

提交

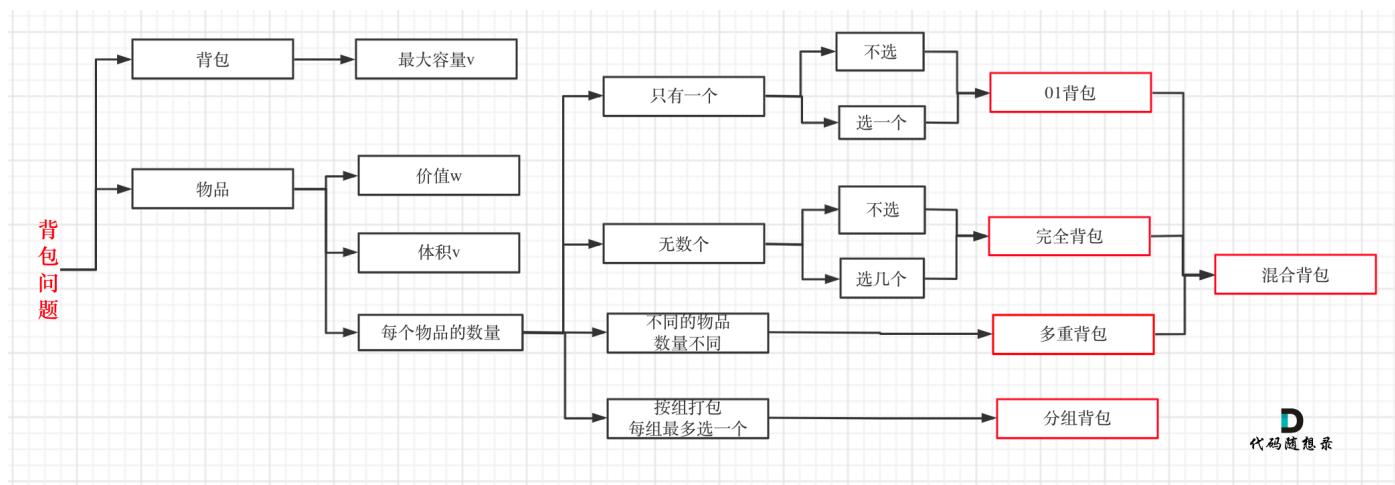


```

1 C, N = map(int, input().split())
2 dp = [0] * (C + 1)
3 w = list(map(int, input().split()))
4 v = list(map(int, input().split()))
5 k = list(map(int, input().split()))
6 print(w, v, k)
7 for i in range(N):
8     for j in range(C, w[i] - 1, -1):
9         for m in range(1, k[i] + 1, 1):
10            if j - m * w[i] >= 0:
11                dp[j] = max(dp[j], dp[j - m * w[i]] + m * v[i])
12            else:
13                break
14 print(dp[C])
15
16     解法同01背包，只是在这里要遍历一下数量，相当于将所有物品的所有数量摊开
17     就是个01背包问题
18     (这样写a不了，也不是时间、内存超，具体原因看卡码网，掌握思路就好)
19

```

6.10 背包问题总结



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

- **递推公式**

类型	递推公式
问能否能装满背包 (或者最多装多少)	$dp[j] = \max(dp[j], dp[j - \text{nums}[i]] + \text{nums}[i])$
问装满背包有几种方法	$dp[j] += dp[j - \text{nums}[i]]$
问背包装满最大价值	$dp[j] = \max(dp[j], dp[j - \text{weight}[i]] + \text{value}[i])$
问装满背包所有物品的最小个数	$dp[j] = \min(dp[j - \text{coins}[i]] + 1, dp[j])$

- 遍历顺序

- 01背包：一维dp数组01背包只能先遍历物品再遍历背包容量，且第二层for循环是**从大到小遍历**
- 完全背包：
 - 求组合数就是外层**for**循环遍历物品，内层**for**遍历背包
 - 求排列数就是外层**for**遍历背包，内层**for**循环遍历物品
 - 求最小数，那么两层for循环的先后顺序就无所谓

6.11 打家劫舍 (I、II、III)

- 198 (I)

198. 打家劫舍

已解答 

中等

相关标签

相关企业

A₂

你是一个专业的小偷，计划偷窃沿街的房屋。每间房内都藏有一定的现金，影响你偷窃的唯一制约因素就是相邻的房屋装有相互连通的防盗系统，如果两间相邻的房屋在同一晚上被小偷闯入，系统会自动报警。

给定一个代表每个房屋存放金额的非负整数数组，计算你 不触动警报装置的情况下，一夜之内能够偷窃到的最高金额。

示例 1：

输入： [1,2,3,1]

输出： 4

解释： 偷窃 1 号房屋（金额 = 1），然后偷窃 3 号房屋（金额 = 3）。

偷窃到的最高金额 = 1 + 3 = 4。

示例 2：

输入： [2,7,9,3,1]

输出： 12

解释： 偷窃 1 号房屋（金额 = 2），偷窃 3 号房屋（金额 = 9），接着偷窃 5 号房屋（金额 = 1）。

偷窃到的最高金额 = 2 + 9 + 1 = 12。

提示：

- `1 <= nums.length <= 100`

- `[0 <= nums[i] <= 400]`

</> 代码

Python3 ✓ 智能模式

☰ ⌂ ⌂ ⌂

```
1 class Solution:
2     def rob(self, nums: List[int]) -> int:
3         n = len(nums)
4         if n == 1:
5             return nums[0]
6         dp = [0] * n
7         dp[0] = nums[0]
8         dp[1] = max(nums[0], nums[1])
9         for i in range(2, n):
10            dp[i] = max(dp[i - 2] + nums[i], dp[i - 1])
11
return dp[n - 1]
```

dp[i]: 前*i*家获取的最大金额 (有第0家, 当然也可以让dp长度为n+1, 相应修改即可)

递推公式: $dp[i] = \max(dp[i-2]+nums[i], dp[i-1])$, 第*i*家偷, 就是 $dp[i-2]+nums[i]$, 第*i*家不偷就是 $dp[i-1]$

已存储至本地

行 9, 列 25

- 213 (II)

213. 打家劫舍 II

已解答

中等 相关标签 相关企业 提示 A*

你是一个专业的小偷，计划偷窃沿街的房屋，每间房内都藏有一定的现金。这个地方所有的房屋都 **围成一圈**，这意味着第一个房屋和最后一个房屋是紧挨着的。同时，相邻的房屋装有相互连通的防盗系统，如果两间相邻的房屋在同一晚上被小偷闯入，系统会自动报警。

给定一个代表每个房屋存放金额的非负整数数组，计算你 **在不触动警报装置的情况下**，今晚能够偷窃到的最高金额。

示例 1：

输入: `nums = [2,3,2]`

输出: 3

解释: 你不能先偷窃 1 号房屋 (金额 = 2) ，然后偷窃 3 号房屋 (金额 = 2) ，因为他们是相邻的。

示例 2：

输入: `nums = [1,2,3,1]`

输出: 4

解释: 你可以先偷窃 1 号房屋 (金额 = 1) ，然后偷窃 3 号房屋 (金额 = 3) 。

偷窃到的最高金额 = 1 + 3 = 4 。

示例 3：

输入: `nums = [1,2,3]`

输出: 3

提示:

- `1 <= nums.length <= 100`
- `0 <= nums[i] <= 1000`

</> 代码

Python3 ▾ 智能模式

↶ ⌂ ⌂ ⌂

```
1 class Solution:          16 19 1, 对于2, 不考虑首考虑尾, 6 19 1最大是6+9, 没有尾的1
2     def rob(self, nums: List[int]) -> int:
3         if len(nums) == 0:  因为闭环, 所以无非三种情况
4             return 0          1. 不考虑首尾元素
5         if len(nums) == 1:  2. 不考虑首元素
6             return nums[0]    3. 不考虑尾元素
7             此处考虑 ≠ 劫这家, 且23包涵1, 所以仅考虑23情况即可
8         result1 = self.robRange(nums, 0, len(nums) - 2) # 情况二
9         result2 = self.robRange(nums, 1, len(nums) - 1) # 情况三
10        return max(result1, result2)
11
12    # 198.打家劫舍的逻辑
13    def robRange(self, nums: List[int], start: int, end: int) -> int:
14        if end == start:
15            return nums[start]
16
17        prev_max = nums[start]
18        curr_max = max(nums[start], nums[start + 1])
19
20        for i in range(start + 2, end + 1):
21            temp = curr_max
22            curr_max = max(prev_max + nums[i], curr_max) 只用两个数就可以, 不需要
23            prev_max = temp                                dp数组
24
25        return curr_max
```

已存储至本地

行 24, 列 24

- 337 (III)

337. 打家劫舍 III

已解答

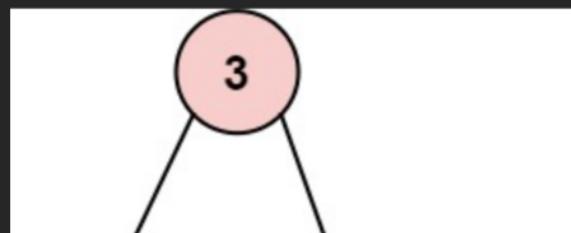
中等 相关标签 相关企业 A₂

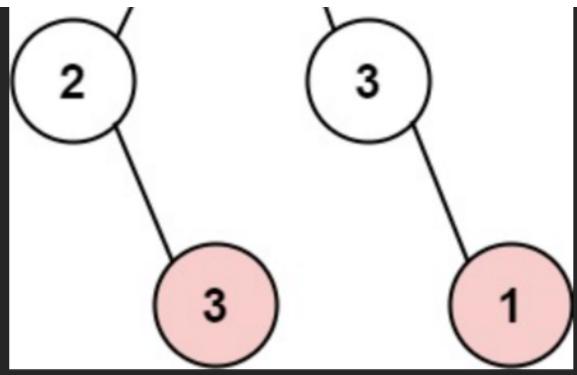
小偷又发现了一个新的可行窃的地区。这个地区只有一个入口，我们称之为 `root`。

除了 `root` 之外，每栋房子有且只有一个“父”房子与之相连。一番侦察之后，聪明的小偷意识到“这个地方的所有房屋的排列类似于一棵二叉树”。如果 两个直接相连的房子在同一天晚上被打劫，房屋将自动报警。

给定二叉树的 `root`。返回 在不触动警报的情况下，小偷能够盗取的最高金额。

示例 1:



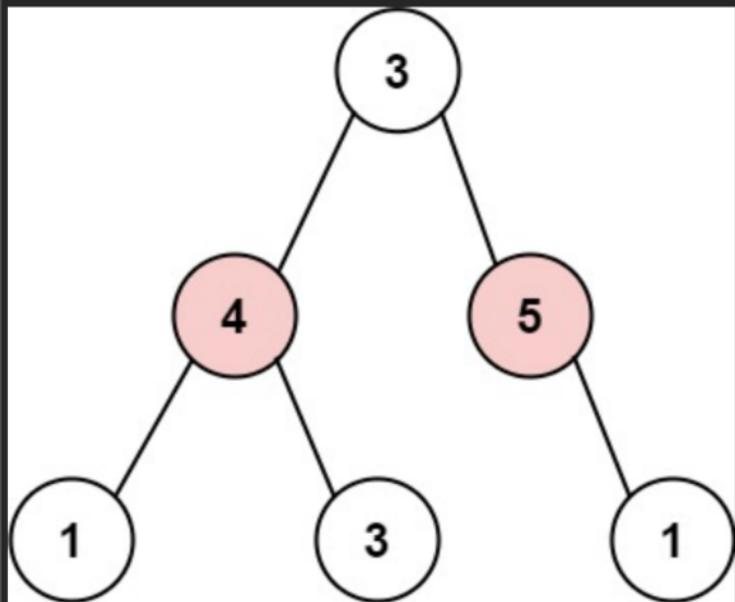


输入: root = [3,2,3,null,3,null,1]

输出: 7

解释: 小偷一晚能够盗取的最高金额 $3 + 3 + 1 = 7$

示例 2:



输入: root = [3,4,5,1,3,null,1]

输出: 9

解释: 小偷一晚能够盗取的最高金额 $4 + 5 = 9$

提示:

- 树的节点数在 $[1, 10^4]$ 范围内

- $0 \leq \text{Node.val} \leq 10^4$

</> 代码

Python3 智能模式

撤销 重置 退出

```
1 # Definition for a binary tree node.
2 # class TreeNode:
3 #     def __init__(self, val=0, left=None, right=None):
4 #         self.val = val
5 #         self.left = left
6 #         self.right = right
7 class Solution:
8     def rob(self, root: Optional[TreeNode]) -> int:
9         return max(self.travel(root)) 递归解决
10
11     def travel(self, node): 后序遍历左右中
12
13         if not node:
14             return (0, 0)
15
16         left = self.travel(node.left)
17         right = self.travel(node.right)
18
19         # 不偷当前节点
20         val1 = max(left[0], left[1]) + max(right[0], right[1])
21         # 偷当前节点
22         val2 = node.val + left[0] + right[0]
23
24         return (val1, val2)
```

已存储至本地

行 1, 列 1

6.12 买卖股票的最佳时机 (121、122、123、188、309、714)

- 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 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

188. 买卖股票的最佳时机 IV

已解答

困难 相关标签 相关企业 A_文

给你一个整数数组 `prices` 和一个整数 `k`，其中 `prices[i]` 是某支给定的股票在第 `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 `k` 笔交易。也就是说，你最多可以买 `k` 次，卖 `k` 次。

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: `k = 2, prices = [2,4,1]`

输出: 2

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

示例 2：

输入: `k = 2, prices = [3,2,6,5,0,3]`

输出: 7

解释: 在第 2 天（股票价格 = 2）的时候买入，在第 3 天（股票价格 = 6）的时候卖出，这笔交易所能获得利润 = 6-2 = 4。

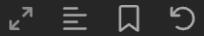
随后，在第 5 天（股票价格 = 0）的时候买入，在第 6 天（股票价格 = 3）的时候卖出，这笔交易所能获得利润 = 3-0 = 3。

提示:

- `1 <= k <= 100`
- `1 <= prices.length <= 1000`
- `0 <= prices[i] <= 1000`

</> 代码

Python3 智能模式



```
1 class Solution:
2     def maxProfit(self, k: int, prices: List[int]) -> int:
3         lens = len(prices)
4         dp = [[0] * (2*k+1) for _ in range(lens)]
5         for j in range(1, 2*k+1, 2):
6             dp[0][j] = -prices[0]
7             只是将2次扩展为k次，此路同上题
8         for i in range(1, lens):
9             for j in range(1, 2*k+1, 2):
10                 dp[i][j] = max(dp[i-1][j], dp[i-1][j-1] - prices[i])
11                 dp[i][j+1] = max(dp[i-1][j+1], dp[i-1][j] + prices[i])
12
13         return dp[-1][2*k]
```

- 309

309. 买卖股票的最佳时机含冷冻期

中等 相关标签 相关企业 A_z

给定一个整数数组 `prices`，其中第 `prices[i]` 表示第 `i` 天的股票价格。

设计一个算法计算出最大利润。在满足以下约束条件下，你可以尽可能地完成更多的交易（多次买卖一支股票）：

- 卖出股票后，你无法在第二天买入股票 (即冷冻期为 1 天)。

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 = [1,2,3,0,2]`

输出: 3

解释: 对应的交易状态为: [买入, 卖出, 冷冻期, 买入, 卖出]

示例 2：

输入: `prices = [1]`

输出: 0

提示:

- `1 <= prices.length <= 5000`
- `0 <= prices[i] <= 1000`

</> 代码

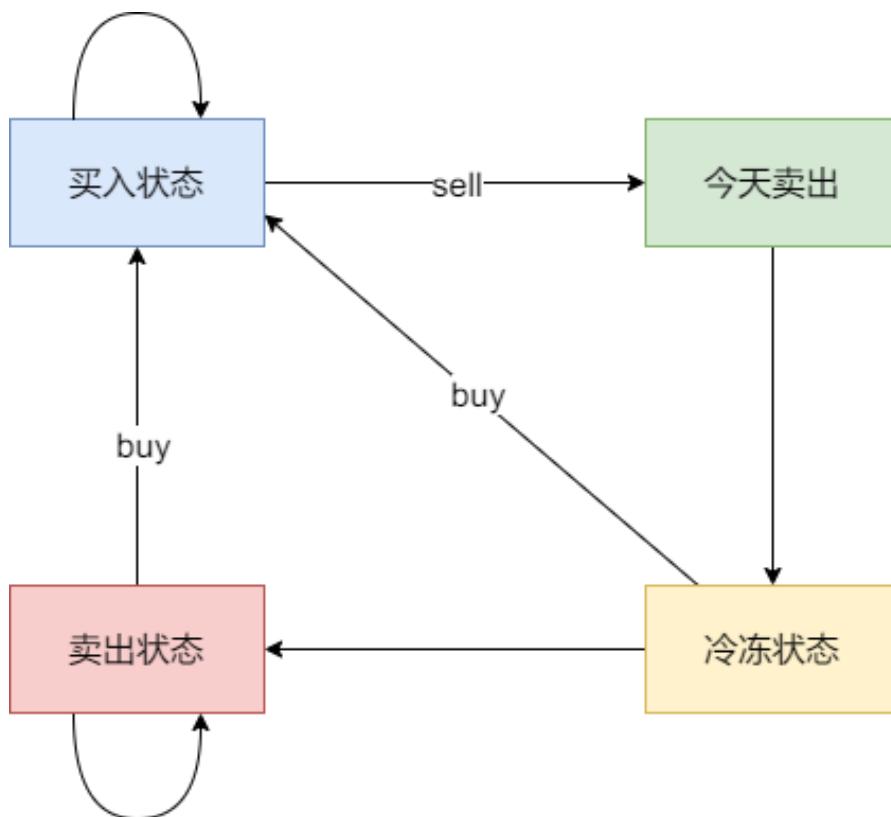
Python3 智能模式

≡ ⌂ ↻ ↺

```
1 class Solution:
2     def maxProfit(self, prices: List[int]) -> int:
3         n = len(prices)
4         dp = [[0] * 4 for _ in range(n)]
5         dp[0][0] = -prices[0]
6         for i in range(1, n):
7             dp[i][0] = max(dp[i-1][0], dp[i-1][1]-prices[i], dp[i-1][3]-prices[i])
8             dp[i][1] = max(dp[i-1][1], dp[i-1][3])
9             dp[i][2] = dp[i-1][0] + prices[i]
10            dp[i][3] = dp[i-1][2]
11        return max(dp[-1][1], dp[-1][2], dp[-1][3])
```

解释：0状态为持有股票状态（买入或之前就买入一直持有），1状态为保持股票卖出状态（冷冻期后），2状态为卖出股票状态（卖出），3状态为冷冻期（卖出后第一天），其关系举例如下

0	0	0	2	3	1	1	1	0	2	3
买入	持有	持有	卖出	冷冻	保持卖出	保持卖出	保持卖出	买入	卖出	冷冻



- `dp[i][0] = max(dp[i-1][0], dp[i-1][1] - prices[i], dp[i-1][3] - prices[i])`: 第*i*天买入, *i*-1天持有 / *i*-1天保持卖出*i*天买入 / *i*-1天冷冻*i*天买入
- `dp[i][1] = max(dp[i-1][1], dp[i-1][3])`: 第*i*天保持卖出, *i*-1天保持卖出 / *i*-1天冷冻
- `dp[i][2] = dp[i-1][0] + prices[i]`: 第*i*天卖出, *i*-1天持有
- `dp[i][3] = dp[i-1][2]`: 第*i*天冷冻, *i*-1天卖出

714. 买卖股票的最佳时机含手续费

已解答

中等 相关标签 相关企业 提示 A_x

给定一个整数数组 `prices`，其中 `prices[i]` 表示第 `i` 天的股票价格；整数 `fee` 代表了交易股票的手续费。

你可以无限次地完成交易，但是你每笔交易都需要付手续费。如果你已经购买了一个股票，在卖出它之前你就不能再继续购买股票了。

返回获得利润的最大值。

注意：这里的一笔交易指买入持有并卖出股票的整个过程，每笔交易你只需要为支付一次手续费。

示例 1：

输入: `prices = [1, 3, 2, 8, 4, 9]`, `fee = 2`

输出: 8

解释: 能够达到的最大利润:

在此处买入 `prices[0] = 1`

在此处卖出 `prices[3] = 8`

在此处买入 `prices[4] = 4`

在此处卖出 `prices[5] = 9`

总利润: $((8 - 1) - 2) + ((9 - 4) - 2) = 8$

示例 2：

输入: `prices = [1,3,7,5,10,3]`, `fee = 3`

输出: 6

提示:

- `1 <= prices.length <= 5 * 104`
- `1 <= prices[i] < 5 * 104`
- `0 <= fee < 5 * 104`

</> 代码



Python3 智能模式



```
1 class Solution:
2     def maxProfit(self, prices: List[int], fee: int) -> int:
3         lens = len(prices)
4         dp = [[0, 0] for _ in range(lens)]
5         # dp[0][0] = - prices[0] - fee           买入/卖出减手续费都能a
6         dp[0][0] = - prices[0]
7         for i in range(1, lens):
8             # dp[i][0] = max(dp[i-1][0], dp[i-1][1] - prices[i] - fee)
9             dp[i][0] = max(dp[i-1][0], dp[i-1][1] - prices[i])
10            # dp[i][1] = max(dp[i-1][1], dp[i-1][0] + prices[i])
11            dp[i][1] = max(dp[i-1][1], dp[i-1][0] + prices[i] - fee)
12
13     return dp[-1][1]
```

(最好还是卖出减手续费，比较好理解)

6.13 子序列问题

- 300 (最长递增子序列)

300. 最长递增子序列

已解答

中等

相关标签

相关企业

A_x

给你一个整数数组 `nums`，找到其中最长严格递增子序列的长度。

子序列 是由数组派生而来的序列，删除（或不删除）数组中的元素而不改变其余元素的顺序。例如，`[3,6,2,7]` 是数组 `[0,3,1,6,2,2,7]` 的**子序列**。

示例 1：

输入: `nums = [10,9,2,5,3,7,101,18]`

输出: 4

解释: 最长递增子序列是 `[2,3,7,101]`，因此长度为 4 。

示例 2：

输入: `nums = [0,1,0,3,2,3]`

输出: 4

示例 3：

输入: `nums = [7,7,7,7,7,7,7]`

输出: 1

提示:

- `1 <= nums.length <= 2500`
- `-104 <= nums[i] <= 104`

`</>` 代码

Python3 智能模式



```
1 class Solution:
2     def lengthOfLIS(self, nums: List[int]) -> int:
3         length = len(nums)
4         dp = [1] * length
5         for i in range(1, length):
6             for j in range(i):
7                 if nums[i] > nums[j]:
8                     dp[i] = max(dp[i], dp[j] + 1)
9         return max(dp)
```

dp[i]: i位置之前包涵i, 以i位置结尾的
最长递增子串长度

- 674 (最长连续递增序列)

674. 最长连续递增序列

已解答

[简单](#) [相关标签](#) [相关企业](#) [Ax](#)

给定一个未经排序的整数数组，找到最长且 **连续递增的子序列**，并返回该序列的长度。

连续递增的子序列 可以由两个下标 l 和 r ($l < r$) 确定，如果对于每个 $l \leq i < r$ ，都有 $\text{nums}[i] < \text{nums}[i + 1]$ ，那么子序列 $[\text{nums}[l], \text{nums}[l + 1], \dots, \text{nums}[r - 1], \text{nums}[r]]$ 就是连续递增子序列。

示例 1：

输入: $\text{nums} = [1,3,5,4,7]$

输出: 3

解释: 最长连续递增序列是 $[1,3,5]$ ，长度为3。

尽管 $[1,3,5,7]$ 也是升序的子序列，但它不是连续的，因为 5 和 7 在原数组里被 4 隔开。

示例 2：

输入: $\text{nums} = [2,2,2,2,2]$

输出: 1

解释: 最长连续递增序列是 $[2]$ ，长度为1。

提示:

- $1 \leq \text{nums.length} \leq 10^4$
- $-10^9 \leq \text{nums}[i] \leq 10^9$

代码

Python3 智能模式



```
1 class Solution:
2     def findLengthOfLCIS(self, nums: List[int]) -> int:
3         length = len(nums)
4         dp = [1] * length
5         for i in range(1, length):
6             if nums[i] > nums[i-1]:
7                 dp[i] = dp[i-1] + 1
8         return max(dp)
```

- 718 (最长重复子数组)

718. 最长重复子数组

中等

相关标签

相关企业

提示

Ax

给两个整数数组 `nums1` 和 `nums2`，返回 两个数组中 公共的、长度最长的子数组的长度。

示例 1：

输入: `nums1 = [1,2,3,2,1]`, `nums2 = [3,2,1,4,7]`

输出: 3

解释: 长度最长的公共子数组是 `[3,2,1]`。

示例 2：

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

输出: 5

提示:

- `1 <= nums1.length, nums2.length <= 1000`

- `0 <= nums1[i], nums2[i] <= 100`

</> 代码

Python3



```
1 class Solution:
2     def findLength(self, nums1: List[int], nums2: List[int]) -> int:
3         length1, length2 = len(nums1), len(nums2)
4         dp = [[0] * (length2 + 1) for _ in range(length1 + 1)]
5         for i in range(1, length1 + 1):
6             for j in range(1, length2 + 1):  
                 if nums1[i-1] == nums2[j-1]:  
                     dp[i][j] = max(dp[i-1][j-1] + 1, dp[i][j])
7                 else:  
                     dp[i][j] = 0
8         return max(max(dp))
```

`dp[i][j]`: `nums1`以*i-1*结尾和`nums2`以*j-1*位置结尾的两个数组最长重复子数组长度（为何*i-1*和*j-1*，看网站解释，这样初始化方便）

注意二维数组求最大值`max(map(max, dp))`！！！

- 1143 (最长公共子序列)

1143. 最长公共子序列

[中等](#) [相关标签](#) [相关企业](#) [提示](#) [A_x](#)

给定两个字符串 `text1` 和 `text2`，返回这两个字符串的最长 公共子序列 的长度。如果不存在 公共子序列，返回 `0`。

一个字符串的 子序列 是指这样一个新的字符串：它是由原字符串在不改变字符的相对顺序的情况下删除某些字符（也可以不删除任何字符）后组成的新字符串。

- 例如，`"ace"` 是 `"abcde"` 的子序列，但 `"aec"` 不是 `"abcde"` 的子序列。

两个字符串的 公共子序列 是这两个字符串所共同拥有的子序列。

示例 1：

输入: `text1 = "abcde"`, `text2 = "ace"`

输出: 3

解释: 最长公共子序列是 `"ace"`，它的长度为 3。

示例 2：

输入: `text1 = "abc"`, `text2 = "abc"`

输出: 3

解释: 最长公共子序列是 `"abc"`，它的长度为 3。

示例 3：

输入: `text1 = "abc"`, `text2 = "def"`

输出: 0

解释: 两个字符串没有公共子序列，返回 0。

提示：

- `1 <= text1.length, text2.length <= 1000`
- `text1` 和 `text2` 仅由小写英文字母组成。

</> 代码

Python3 智能模式

≡ ⌂ ↻ ↺

```
1 class Solution:
2     def longestCommonSubsequence(self, text1: str, text2: str) -> int:
3         dp = [[0] * (len(text2) + 1) for _ in range(len(text1) + 1)]
4         for i in range(1, len(text1) + 1):
5             for j in range(1, len(text2) + 1):
6                 if text1[i - 1] == text2[j - 1]:
7                     dp[i][j] = dp[i - 1][j - 1] + 1    主要理解这个递推公式
8                 else:
9                     dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])
10
11     return dp[-1][-1]
```

至于此处为什么是 `dp[i][j] = max(dp[i-1][j], dp[i][j-1])`，我理解的是如果 `text1[i-1] == text2[j-1]`，相等时用到了这两个位置上的字母，所以取的是 `dp[i][j] = dp[i-1][j-1] + 1`；如果不相等，则没有用到这两个位置的字母，那么递推公式取max时就可以用到这两个位置保存的信息，相当于用到了这两个位置的字母

- 1035 (不相交的线)

1035. 不相交的线

中等

相关标签

相关企业

提示

Ax

在两条独立的水平线上按给定的顺序写下 `nums1` 和 `nums2` 中的整数。

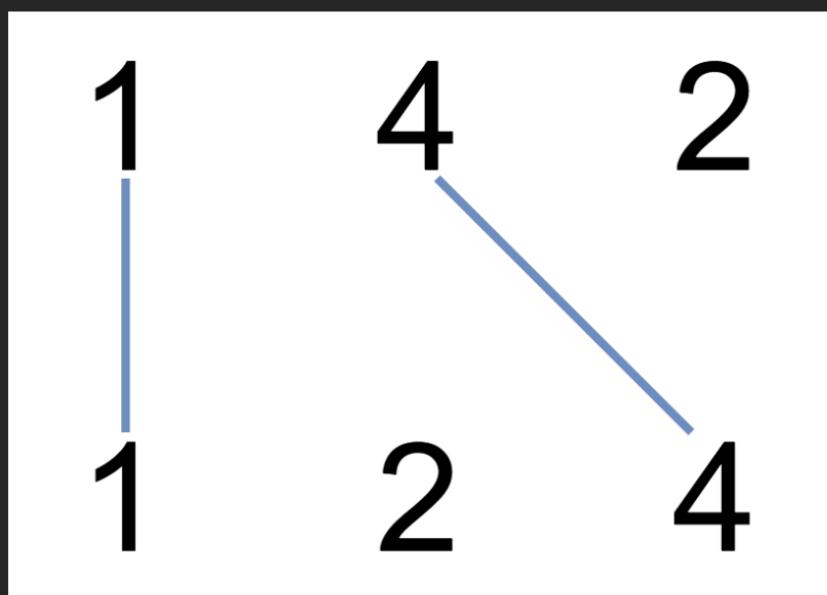
现在，可以绘制一些连接两个数字 `nums1[i]` 和 `nums2[j]` 的直线，这些直线需要同时满足：

- `nums1[i] == nums2[j]`
- 且绘制的直线不与任何其他连线（非水平线）相交。

请注意，连线即使在端点也不能相交：每个数字只能属于一条连线。

以这种方法绘制线条，并返回可以绘制的最大连线数。

示例 1：



输入: `nums1 = [1,4,2]`, `nums2 = [1,2,4]`

输出: 2

解释: 可以画出两条不交叉的线, 如上图所示。

但无法画出第三条不相交的直线, 因为从 `nums1[1]=4` 到 `nums2[2]=4` 的直线将与从 `nums1[2]=2` 到 `nums2[1]=2` 的直线相交。

示例 2：

输入: `nums1 = [2,5,1,2,5]`, `nums2 = [10,5,2,1,5,2]`

输出: 3

示例 3：

输入: `nums1 = [1,3,7,1,7,5]`, `nums2 = [1,9,2,5,1]`

输出: 2

</> 代码

Python3 智能模式

≡ ⌂ ↻ ↺

```
1 class Solution:
2     def maxUncrossedLines(self, nums1: List[int], nums2: List[int]) -> int:
3         dp = [[0] * (len(nums2) + 1) for _ in range(len(nums1) + 1)]
4         for i in range(1, len(nums1) + 1):
5             for j in range(1, len(nums2) + 1):
6                 if nums1[i-1] == nums2[j-1]:
7                     dp[i][j] = dp[i-1][j-1] + 1
8                 else:
9                     dp[i][j] = max(dp[i-1][j], dp[i][j-1])
10        return dp[-1][-1]
```

其实就是找`nums1`的`0~i-1`和`nums2`的`0~j-1`的最长公共子序列
相对顺序不变即可

已存储至本地

行 9, 列 59

- 53

53. 最大子数组和

已解答

中等 相关标签 相关企业 A₂

给你一个整数数组 `nums`，请你找出一个具有最大和的连续子数组（子数组最少包含一个元素），返回其最大和。

子数组是数组中的一个连续部分。

示例 1:

输入: `nums = [-2,1,-3,4,-1,2,1,-5,4]`

输出: 6

解释: 连续子数组 `[4,-1,2,1]` 的和最大，为 6 。

示例 2:

输入: `nums = [1]`

输出: 1

示例 3:

输入: `nums = [5,4,-1,7,8]`

输出: 23

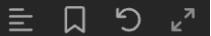
提示:

- `1 <= nums.length <= 105`
- `-104 <= nums[i] <= 104`

</> 代码



Python3 智能模式



```
1 class Solution:
2     def maxSubArray(self, nums: List[int]) -> int:
3         dp = [0] * len(nums)
4         dp[0] = nums[0]
5         for i in range(1, len(nums)):
6             if nums[i] + dp[i-1] > nums[i]:
7                 dp[i] = nums[i] + dp[i-1]
8             else:
9                 dp[i] = nums[i]
10    return max(dp)
```

dp[i]: 包括位置i的i及i之前的连续子串最大和
因为是连续，如果nums[i]大，让dp[i]等于nums[i]即可

- 392 (判断子序列)

392. 判断子序列

[简单](#) [相关标签](#) [相关企业](#) [Ax](#)

给定字符串 s 和 t，判断 s 是否为 t 的子序列。

字符串的一个子序列是原始字符串删除一些（也可以不删除）字符而不改变剩余字符相对位置形成的新字符串。（例如，“ace”是“abcde”的一个子序列，而“aec”不是）。

进阶：

如果有大量输入的 S，称作 S1, S2, … , Sk 其中 k >= 10亿，你需要依次检查它们是否为 T 的子序列。在这种情况下，你会怎样改变代码？

示例 1：

输入: s = "abc", t = "ahbgdc"
输出: true

示例 2：

输入: s = "axc", t = "ahbgdc"
输出: false

提示：

- $0 \leq s.length \leq 100$
- $0 \leq t.length \leq 10^4$
- 两个字符串都只由小写字符组成。

</> 代码

Python3 ▾ 🔒 智能模式



```
1 class Solution:
2     def isSubsequence(self, s: str, t: str) -> bool:
3         sl = len(s)
4         tl = len(t)
5         dp = [[0] * (tl + 1) for _ in range(sl + 1)]
6         for i in range(1, sl + 1):
7             for j in range(1, tl + 1):
8                 if s[i-1] == t[j-1]:
9                     dp[i][j] = dp[i-1][j-1] + 1
10                else:
11                    dp[i][j] = dp[i][j-1] 不取max最好，因为s需要完整的
12                    # dp[i][j] = max(dp[i-1][j], dp[i][j-1])
13        return dp[-1][-1] == len(s)
```

- 115 (不同的子序列)

115. 不同的子序列

已解答

困难 相关标签 相关企业 A文

给你两个字符串 s 和 t ，统计并返回在 s 的 子序列 中 t 出现的个数，结果需要对 $10^9 + 7$ 取模。

示例 1：

输入: $s = "rabbbit"$, $t = "rabbit"$

输出: 3

解释:

如下所示，有 3 种可以从 s 中得到 "rabbit" 的方案。

rabbbit

rabbbit

rabbbit

示例 2：

输入: $s = "babgbag"$, $t = "bag"$

输出: 5

解释:

如下所示，有 5 种可以从 s 中得到 "bag" 的方案。

babgbag

babgbag

babgbbag

babgbbag

babgbbag

提示:

- $1 \leq s.length, t.length \leq 1000$

- s 和 t 由英文字母组成

</> 代码

Python3 ▾ 🔒 智能模式

☰ ⌂ ⌄ ↗

```
1 class Solution:
2     def numDistinct(self, s: str, t: str) -> int:
3         sl = len(s)
4         tl = len(t)
5         dp = [[0] * (sl + 1) for _ in range(tl + 1)]
6         for i in range(sl + 1):
7             dp[0][i] = 1
8         for i in range(1, tl + 1):
9             for j in range(1, sl + 1):
10                 if t[i - 1] == s[j - 1]:
11                     dp[i][j] = dp[i - 1][j - 1] + dp[i][j - 1]
12                 else:
13                     dp[i][j] = dp[i][j - 1]
14         return int(dp[-1][-1] % (1e9 + 7))
```

dp[i][j]: t中i-1及之前，s中j-1及之前，s中包涵t的子序列数量

递推公式： $t[i-1] == s[j-1]$ ，分两种情况，一是s用到j-1的字母，则数量为 $dp[i-1][j-1]$ ，二是s用不到j-1的字母，则数量为 $dp[i][j-1]$ ，举例比如t为bag，s为baegg，当求 $dp[3][5]$ 时，可以是baeg，也可以是baegg的第二个g，所以要两者相加

$t[i-1] \neq s[j-1]$ ，则数量直接取 $dp[i][j-1]$ ，因为需要t的所有字符，所以只j-1

初始化： $dp[0][i]$ 指的是t为空字符串，s任意字符串删除任意字符与t相同，即全删，变为空字符串，数量为1

- 583 (两个字符串的删除操作)

583. 两个字符串的删除操作

中等

相关标签

相关企业

A_x

给定两个单词 word1 和 word2，返回使得 word1 和 word2 相同所需的最小步数。

每步 可以删除任意一个字符串中的一个字符。

示例 1：

输入: word1 = "sea", word2 = "eat"

输出: 2

解释: 第一步将 "sea" 变为 "ea" , 第二步将 "eat "变为 "ea"

示例 2：

输入: word1 = "leetcode", word2 = "etco"

输出: 4

提示:

- $1 \leq \text{word1.length}, \text{word2.length} \leq 500$

- word1 和 word2 只包含小写英文字母

</> 代码

Python3



```
1 class Solution:
2     def minDistance(self, word1: str, word2: str) -> int:
3         wl1 = len(word1)
4         wl2 = len(word2)
5         dp = [[0] * (wl2 + 1) for _ in range(wl1 + 1)]
6         for i in range(1, wl1 + 1):
7             for j in range(1, wl2 + 1):
8                 if word1[i-1] == word2[j-1]:
9                     dp[i][j] = dp[i-1][j-1] + 1
10                else:
11                    dp[i][j] = max(dp[i-1][j], dp[i][j-1])
12
13 return wl1 + wl2 - 2 * dp[-1][-1]
```

将问题转化为求word1和word2中最长公共子串长度

- 72 (编辑距离)

72. 编辑距离

已解答

中等 相关标签 相关企业 A₂

给你两个单词 `word1` 和 `word2`，请返回将 `word1` 转换成 `word2` 所使用的最少操作数。

你可以对一个单词进行如下三种操作：

- 插入一个字符
- 删 除一个字符
- 替换一个字符

示例 1：

输入: `word1 = "horse"`, `word2 = "ros"`

输出: 3

解释:

`horse` -> `orse` (将 'h' 替换为 'r')

`orse` -> `ose` (删除 'r')

`ose` -> `os` (删除 'e')

示例 2：

输入: `word1 = "intention"`, `word2 = "execution"`

输出: 5

解释:

`intention` -> `inention` (删除 't')

`inention` -> `enention` (将 'i' 替换为 'e')

`enention` -> `exention` (将 'n' 替换为 'x')

`exention` -> `exection` (将 'n' 替换为 'c')

`exection` -> `execution` (插入 'u')

提示:

- `0 <= word1.length, word2.length <= 500`

- `word1` 和 `word2` 由小写英文字母组成

</> 代码

Python3 智能模式

dp[i][j]: word1的0~i-1位置与word2的0~j-1

位置最小编辑距离

```
1 class Solution:
2     def minDistance(self, word1: str, word2: str) -> int:
3         wl1 = len(word1)
4         wl2 = len(word2)
5         dp = [[0] * (wl2 + 1) for _ in range(wl1 + 1)]
6         for i in range(wl1 + 1):
7             dp[i][0] = i
8         for i in range(wl2 + 1):
9             dp[0][i] = i
10        for i in range(1, wl1 + 1):
11            for j in range(1, wl2 + 1):
12                if word1[i-1] == word2[j-1]:
13                    dp[i][j] = dp[i-1][j-1] 相等时直接等于上一个位置
14                else:
15                    dp[i][j] = min(dp[i-1][j], dp[i][j-1], dp[i-1]
16 [j-1]) + 1      word1增加一个 (也即word2减少一个) ↑
                           word2增加一个 (也即word1减少一个) ↑
                           word1的i-1位置替换一个
16         return dp[-1][-1]
```

已存储至本地

行 16, 列 23

• 647

647. 回文子串

中等

相关标签

相关企业

提示

Ax

给你一个字符串 `s`，请你统计并返回这个字符串中 回文子串 的数目。

回文字符串 是正着读和倒过来读一样的字符串。

子字符串 是字符串中的由连续字符组成的一个序列。

具有不同开始位置或结束位置的子串，即使是由相同的字符组成，也会被视作不同的子串。

示例 1：

输入: `s = "abc"`

输出: 3

解释: 三个回文子串: "a", "b", "c"

示例 2：

输入: `s = "aaa"`

输出: 6

解释: 6个回文子串: "a", "a", "a", "aa", "aa", "aaa"

提示：

- `1 <= s.length <= 1000`
- `s` 由小写英文字母组成

</> 代码

Python3 智能模式



```
1 class Solution:      dp[i][j]: 位置i-1~j-1是否为回文串
2     def countSubstrings(self, s: str) -> int:
3         length = len(s)
4         dp = [[0] * (length + 1) for _ in range(length + 1)]
5
6         for i in range(length, 0, -1):      因为dp[i][j]求解依据dp[i+1][j-1]
7             for j in range(i, length + 1):  所以要从下往上, 从左往右遍历
8                 if s[i-1] == s[j-1]:          左→右 右→左都可以
9                     if j - i <= 2:
10                         dp[i][j] = 1
11                     else:
12                         dp[i][j] = dp[i+1][j-1]
13
14         return sum(map(sum, dp))
```

- 516

516. 最长回文子序列

中等 相关标签 相关企业 A_x

给你一个字符串 `s`，找出其中最长的回文子序列，并返回该序列的长度。

子序列定义为：不改变剩余字符顺序的情况下，删除某些字符或者不删除任何字符形成的一个序列。

示例 1：

输入：`s = "bbbab"`

输出：4

解释：一个可能的最长回文子序列为 "bbbb" 。

示例 2：

输入：`s = "cbbd"`

输出：2

解释：一个可能的最长回文子序列为 "bb" 。

提示：

- `1 <= s.length <= 1000`

- `s` 仅由小写英文字母组成

</> 代码

Python3 智能模式

三 口 ↪ ↮ ↵

dp[i][j]: 位置i-1~j-1最长回文子串长

```
1 class Solution:
2     def longestPalindromeSubseq(self, s: str) -> int:
3         length = len(s)
4         dp = [[0] * (length + 1) for _ in range(length + 1)]
5         for i in range(length, 0, -1):
6             for j in range(i, length + 1, 1):
7                 if j == i:
8                     dp[i][j] = 1
9                 elif j - i == 1:
10                     if s[i-1] == s[j-1]:
11                         dp[i][j] = 2
12                     else:
13                         dp[i][j] = 1 注意此处赋值1
14                 elif s[i-1] == s[j-1]:
15                     dp[i][j] = dp[i+1][j-1] + 2
16                 else:
17                     dp[i][j] = max(dp[i+1][j], dp[i][j-1])
18         return max(max, dp)
```

同时算上位置i-1和j-1对
结果没有增加，单独考
虑位置i和位置j-2

七、贪心算法

- 455

455. 分发饼干

已解答

[简单](#) [相关标签](#) [相关企业](#) [A₂](#)

假设你是一位很棒的家长，想要给你的孩子们一些小饼干。但是，每个孩子最多只能给一块饼干。

对每个孩子 i ，都有一个胃口值 $g[i]$ ，这是能让孩子们满足胃口的饼干的最小尺寸；并且每块饼干 j ，都有一个尺寸 $s[j]$ 。如果 $s[j] \geq g[i]$ ，我们可以将这个饼干 j 分配给孩子 i ，这个孩子会得到满足。你的目标是尽可能满足越多数量的孩子，并输出这个最大数值。

示例 1：

输入: $g = [1, 2, 3]$, $s = [1, 1]$

输出: 1

解释:

你有三个孩子和两块小饼干，3个孩子的胃口值分别是：1, 2, 3。

虽然你有两块小饼干，由于他们的尺寸都是1，你只能让胃口值是1的孩子满足。

所以你应该输出1。

示例 2：

输入: $g = [1, 2]$, $s = [1, 2, 3]$

输出: 2

解释:

你有两个孩子和三块小饼干，2个孩子的胃口值分别是1, 2。

你拥有的饼干数量和尺寸都足以让所有孩子满足。

所以你应该输出2.

提示:

- $1 \leq g.length \leq 3 * 10^4$
- $0 \leq s.length \leq 3 * 10^4$
- $1 \leq g[i], s[j] \leq 2^{31} - 1$

</> 代码

Python3 ▾ 智能模式

☰ ⌂ ⌄ ↗

```
1 class Solution:
2     def findContentChildren(self, g: List[int], s: List[int]) -> int:
3         g.sort()
4         s.sort()
5         index = 0
6         result = 0
7         for i in range(len(g)):
8             while index < len(s):
9                 if g[i] <= s[index]:
10                     result += 1
11                     index += 1
12                     break
13             index += 1
14         return result
```

核心就是大饼干给大胃口 / 小饼干
给小胃口，遍历顺序只要逻辑对就能 a

- 376

376. 摆动序列

已解答 

[中等](#) [相关标签](#) [相关企业](#) A_x

如果连续数字之间的差严格地在正数和负数之间交替，则数字序列称为 **摆动序列**。第一个差（如果存在的話）可能是正数或负数。仅有一个元素或者含两个不等元素的序列也视作摆动序列。

- 例如，`[1, 7, 4, 9, 2, 5]` 是一个 **摆动序列**，因为差值 `(6, -3, 5, -7, 3)` 是正负交替出现的。
- 相反，`[1, 4, 7, 2, 5]` 和 `[1, 7, 4, 5, 5]` 不是摆动序列，第一个序列是因为它的前两个差值都是正数，第二个序列是因为它的最后一个差值为零。

子序列 可以通过从原始序列中删除一些（也可以不删除）元素来获得，剩下的元素保持其原始顺序。

给你一个整数数组 `nums`，返回 `nums` 中作为 **摆动序列** 的 **最长子序列的长度**。

示例 1：

输入: `nums = [1,7,4,9,2,5]`

输出: 6

解释: 整个序列均为摆动序列，各元素之间的差值为 `(6, -3, 5, -7, 3)`。

示例 2：

输入: `nums = [1,17,5,10,13,15,10,5,16,8]`

输出: 7

解释: 这个序列包含几个长度为 7 摆动序列。

其中一个是 `[1, 17, 5, 10, 13, 15, 10]`，各元素之间的差值为 `(16, -7, 3, -3, 6, -8)`。

示例 3：

输入: `nums = [1,2,3,4,5,6,7,8,9]`

输出: 2

提示：

- `1 <= nums.length <= 1000`
- `|0 <= nums[i] <= 1000`

</> 代码

Python3 ▾ 智能模式

☰ ⌂ ⌄ ↗

```
1 class Solution:
2     def wiggleMaxLength(self, nums: List[int]) -> int:
3         if len(nums) == 1:
4             return 1
5         if len(nums) == 2 and nums[0] != nums[1]:
6             return 2
7         # 找到开始上下坡的第一个数
8         j = 0
9         while j < len(nums) - 1 and nums[j] == nums[j+1]:
10            j += 1
11         # 数组所有数相等且长度大于等于2, 不是摆动序列
12         if j == len(nums) - 1:
13             return 1
14         # 判断是上坡还是下坡, 1上坡, 0下坡
15         flag = 1 if nums[j+1] > nums[j] else -1
16         result = 2
17         j += 1
18         for i in range(j, len(nums) - 1):
19             if (nums[i+1] - nums[i]) * flag < 0:
20                 flag = flag * (-1)
21                 result += 1
22         return result
```

把数组变成上下坡形式
把所有坡间数字删掉，就是最长摆动序列长度
需要注意的是一些边界条件
判断

所有数一样的数组返回1，这是用例测试出来的

• 53

53. 最大子数组和

已解答

中等 相关标签 相关企业 A_x

给你一个整数数组 `nums`，请你找出一个具有最大和的连续子数组（子数组最少包含一个元素），返回其最大和。

子数组是数组中的一个连续部分。

示例 1：

输入: `nums = [-2,1,-3,4,-1,2,1,-5,4]`

输出: 6

解释: 连续子数组 `[4,-1,2,1]` 的和最大，为 6。

示例 2：

输入: `nums = [1]`

输出: 1

示例 3：

输入: `nums = [5,4,-1,7,8]`

输出: 23

提示：

- `1 <= nums.length <= 105`
- `-104 <= nums[i] <= 104`

</> 代码

Python3 智能模式



```
15 ...
16 贪心
17 ...
18 class Solution:
19     def maxSubArray(self, nums: List[int]) -> int:
20         result = float('-inf')
21         count = 0
22         for i in range(len(nums)):
23             count += nums[i]
24             if count > result:
25                 result = count
26             if count < 0:
27                 count = 0
28         return result
```

维护一个count：到当前数的连续子数组
最大和
当count<0时，贪心贪的地方在于此，重
置count=0，从下一个数继续判断
此处不是遇到负数就重置count，而是加
和为负数时才重置，比如4 -1 2

122. 买卖股票的最佳时机 II

难度 中等 山 2122 ★ ⚡ 文档 📢

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：

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

55. 跳跃游戏

已解答 ✓

[中等](#) [!\[\]\(4bc69453e9e94d8b7104de7b173c6ea6_img.jpg\) 相关标签](#) [!\[\]\(6e22c3682f4936b6753d47b10b79c50a_img.jpg\) 相关企业](#) [A股](#)

给你一个非负整数数组 `nums`，你最初位于数组的第一个下标。数组中的每个元素代表你在该位置可以跳跃的最大长度。

判断你是否能够到达最后一个下标，如果可以，返回 `true`；否则，返回 `false`。

示例 1：

输入: nums = [2,3,1,1,4]

输出: true

解释: 可以先跳 1 步, 从下标 0 到达下标 1, 然后再从下标 1 跳 3 步到达最后一个下标。

示例 2：

输入: nums = [3,2,1,0,4]

输出: false

解释：无论怎样，总会到达下标为 3 的位置。但该下标的最大跳跃长度是 0，所以永远不可能到达最后一个下标。

提示：

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

</> 代码

Python3 ✓ 智能模式

≡ ⌂ ↺ ↻

```
1 class Solution:
2     def canJump(self, nums: List[int]) -> bool:
3         maxloc = 0
4         if nums[0] == 0 and len(nums) == 1:
5             return True
6         for i in range(len(nums)):
7             if i <= maxloc:
8                 maxloc = max(maxloc, nums[i] + i)
9             if maxloc >= len(nums) - 1:
10                return True
11        return False
```

45. 跳跃游戏 II

中等 相关标签 相关企业 A_文

给定一个长度为 n 的 0 索引整数数组 nums 。初始位置为 $\text{nums}[0]$ 。

每个元素 $\text{nums}[i]$ 表示从索引 i 向前跳转的最大长度。换句话说，如果你在 $\text{nums}[i]$ 处，你可以跳转到任意 $\text{nums}[i + j]$ 处：

- $0 \leq j \leq \text{nums}[i]$
- $i + j < n$

返回到达 $\text{nums}[n - 1]$ 的最小跳跃次数。生成的测试用例可以到达 $\text{nums}[n - 1]$ 。

示例 1：

输入： $\text{nums} = [2, 3, 1, 1, 4]$

输出： 2

解释： 跳到最后一个位置的最小跳跃数是 2。

从下标为 0 跳到下标为 1 的位置，跳 1 步，然后跳 3 步到达数组的最后一个位置。

示例 2：

输入： $\text{nums} = [2, 3, 0, 1, 4]$

输出： 2

提示：

- $1 \leq \text{nums.length} \leq 10^4$
- $0 \leq \text{nums}[i] \leq 1000$
- 题目保证可以到达 $\text{nums}[n-1]$

- 45

45. 跳跃游戏 II

中等 相关标签 相关企业 A_文

给定一个长度为 n 的 0 索引整数数组 nums 。初始位置为 $\text{nums}[0]$ 。

每个元素 $\text{nums}[i]$ 表示从索引 i 向前跳转的最大长度。换句话说，如果你在 $\text{nums}[i]$ 处，你可以跳转到任意 $\text{nums}[i + j]$ 处：

- $0 \leq j \leq \text{nums}[i]$
- $i + j < n$

返回到达 $\text{nums}[n - 1]$ 的最小跳跃次数。生成的测试用例可以到达 $\text{nums}[n - 1]$ 。

示例 1：

输入： $\text{nums} = [2, 3, 1, 1, 4]$

输出： 2

解释： 跳到最后一个位置的最小跳跃数是 2。

从下标为 0 跳到下标为 1 的位置，跳 1 步，然后跳 3 步到达数组的最后一个位置。

示例 2：

输入： $\text{nums} = [2, 3, 0, 1, 4]$

输出： 2

提示：

- $1 \leq \text{nums.length} \leq 10^4$
- $0 \leq \text{nums}[i] \leq 1000$
- 题目保证可以到达 $\text{nums}[n-1]$

</> 代码

Python3 智能模式

三 口 5 ↗

```
1 ...
2 0(n2)时间复杂度太高，但思路好理解 min_steps维护到达每个位置的最短步数
3 ...
4 # class Solution:
5 #     def jump(self, nums: List[int]) -> int:
6 #         n = len(nums)
7 #         min_steps = [i for i in range(n)]
8 #         for i in range(n):
9 #             for j in range(i+1, min(i+nums[i]+1, n)):
10 #                 min_steps[j] = min(min_steps[j], min_steps[i] + 1)
11 #         return min_steps[-1]
```

</> 代码

Python3 智能模式

```
13  ...
14  贪心版本
15  ...
16  class Solution:
17      def jump(self, nums):
18          if len(nums) == 1:
19              return 0
20
21          cur_distance = 0 # 当前覆盖最远距离下标
22          ans = 0 # 记录走的最大步数
23          next_distance = 0 # 下一步覆盖最远距离下标
24
25          for i in range(len(nums)):
26              next_distance = max(nums[i] + i, next_distance) # 更新下一步覆盖最远距离下标
27              if i == cur_distance: # 遇到当前覆盖最远距离下标
28                  ans += 1 # 需要走下一步
29                  cur_distance = next_distance # 更新当前覆盖最远距离下标（相当于加油了）
30                  if next_distance >= len(nums) - 1: # 当前覆盖最远距离达到数组末尾，不用再
31                      做ans++操作，直接结束
32
33          return ans
```

这里不需要担心next_distance的起点i已经超出了cur_distance，因为每次循环，如果不触发*i==那个*条件，说明*i<cur_distance*，即next_distance是在当前可跳范围内的下一步最远位置

触发*i==*条件时，就可以把当前范围更新到下一步范围，且步数++，如果*cur_distance*已经 $\geq \text{len}-1$ ，说明当前已覆盖到最后，返回*ans*即可

- 1005

1005. K 次取反后最大化的数组和

简单

相关标签

相关企业

A_文

给你一个整数数组 `nums` 和一个整数 `k`，按以下方法修改该数组：

- 选择某个下标 `i` 并将 `nums[i]` 替换为 `-nums[i]`。

重复这个过程恰好 `k` 次。可以多次选择同一个下标 `i`。

以这种方式修改数组后，返回数组 可能的最大和。

示例 1：

输入: `nums = [4,2,3], k = 1`

输出: 5

解释: 选择下标 1，`nums` 变为 `[4,-2,3]`。

示例 2：

输入: `nums = [3,-1,0,2], k = 3`

输出: 6

解释: 选择下标 (1, 2, 2)，`nums` 变为 `[3,1,0,2]`。

示例 3：

输入: `nums = [2,-3,-1,5,-4], k = 2`

输出: 13

解释: 选择下标 (1, 4)，`nums` 变为 `[2,3,-1,5,4]`。

提示:

- `1 <= nums.length <= 104`
- `-100 <= nums[i] <= 100`
- `1 <= k <= 104`

</> 代码

Python3 智能模式

三 双 ⌂ ↻

```
1 ...
2 自己的思路，排序按照从小到大，和代码随想录贪心思路一致，只是有点复杂，时间慢一点
3 ...
4 # class Solution:
5 #     def largestSumAfterKNegations(self, nums: List[int], k: int) -> int:
6 #         nums.sort() 按照大小排序，小到大
7 #         n = len(nums)
8 #         min_nums = float('inf') 维护一个min_nums，用于反复反转
9 #         ans = 0
10 #         for i in range(n):
11 #             if nums[i] < 0 and k > 0: 当前值小于0且k大于0，反转当前值
12 #                 nums[i] = nums[i] * -1 记录min_nums, k--
13 #                 min_nums = min(min_nums, nums[i])
14 #             k -= 1 当前值等于0或k等于0，直接返回sum即可
15 #             elif nums[i] == 0 or k == 0: 当前值等于0可以反复反转0，且负值已反转完
16 #                 return sum(nums) 当前值大于0，即遍历到的第一个本身大于0的数
17 #             elif nums[i] > 0: 更新min_nums，直接return即可，后边不再遍历
18 #                 min_nums = min(min_nums, nums[i])
19 #                 return sum(nums) - 2 * min_nums if k % 2 == 1 else sum(nums)
20 #             return sum(nums) - 2 * min_nums if k % 2 == 1 else sum(nums)
```

</> 代码

Python3 智能模式

三 双 ⌂ ↻

```
18 # min_nums = min(min_nums, nums[i])
19 # return sum(nums) - 2 * min_nums if k % 2 == 1 else sum(nums)
20 # return sum(nums) - 2 * min_nums if k % 2 == 1 else sum(nums)
21
22 ...
23 代码随想录贪心思路，按照绝对值大小排序！！
24 ...
25 class Solution:
26     def largestSumAfterKNegations(self, A: List[int], K: int) -> int:
27         A.sort(key=lambda x: abs(x), reverse=True) # 第一步：按照绝对值降序排序数组A
28
29         for i in range(len(A)): # 第二步：执行K次取反操作
30             if A[i] < 0 and K > 0:
31                 A[i] *= -1
32                 K -= 1
33
34             if K % 2 == 1: # 第三步：如果K还有剩余次数，将绝对值最小的元素取反
35                 A[-1] *= -1
36
37         result = sum(A) # 第四步：计算数组A的元素和
38         return result
```