

九章算法
www.jiuzhang.com

算法高频冲刺班

30天掌握新题、难题、高频题

讲师：夏天+简单



扫码咨询
课程信息



课程特色

- 1 学会一题多解，不怕考题多变
- 2 算法原理掌握，新题套路秒破
- 3 考前押题冲刺，面试如抄原题
- 4 九章讲师天团，助力大厂上岸

优惠码：30248C

扫码使用加入正式课



算法高频题冲刺班团购优惠码:

30248C

扫码使用加入正式班↓



扫码进群，领课件、回放、
实时面经等超多福利



【2月】冲刺班春招面经分享福
利群



该二维码7天内(2月15日前)有效，重新进入将更新

九章算法面试高频题冲刺班直播课1

算法面试高频知识点与技巧

主讲：夏天



加班主任小佳佳，进班级答疑群，快速获取课程福利♡(´·̊·`)

今天穿这样!



讲师夏天

- 九章算法三部曲直播课讲师
 - 九章算法基础班
 - 九章算法班2022版
 - 九章算法面试高频题冲刺班
- 一对一私教辅导讲师
- 多年算法教学经验，帮助1000+学生拿到心仪 offer ? ? ? ? ?
- 在美工作10+年，供职于多家知名科技公司，现任 Engineering Manager
- 刷题数量1000+
- 面试人数300+



助教团队

- 获得过算法竞赛金奖
- 刷题数量1000+









课程定位

- 帮助求职者完成算法面试准备过程中的“最后一公里”
- 解决新题/难题/高频题/知识点路/技(套)巧(路)

最后一公里（Last kilometer），原意指完成长途跋涉的最后一段里程，被引申为完成一件事情的时候最后的而且是关键性的步骤（通常还说明此步骤充满困难）。

如何高效冲刺最后一公里？

- 按照算法、知识点的考察频率从高到低准备
- 针对性的去找对应面试企业的面经题
- 背诵高频算法模板

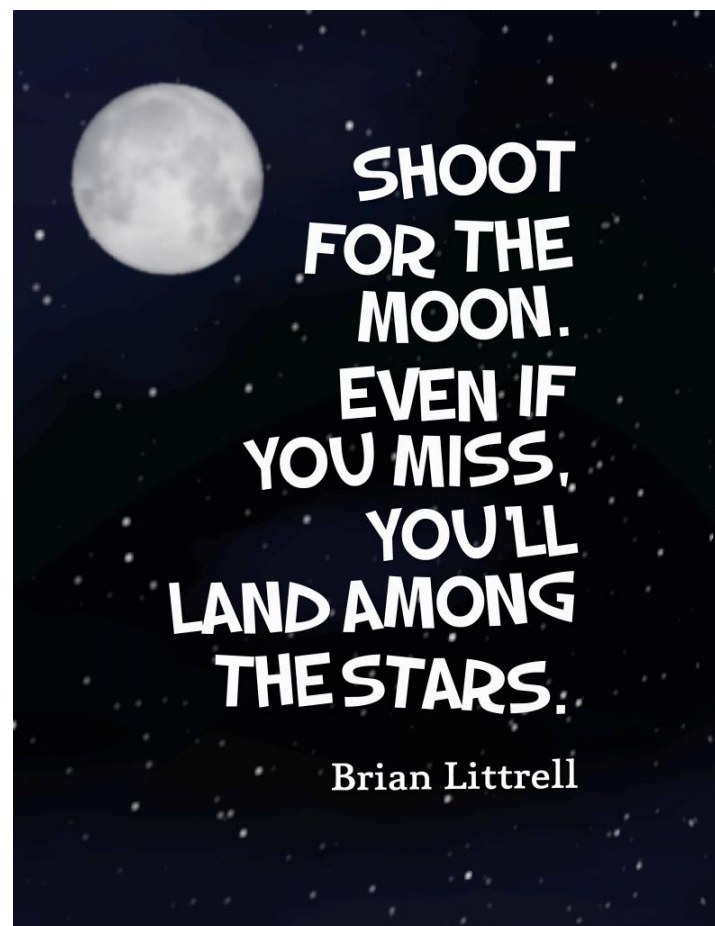
九章金牌课程**三部曲**，环环相扣，循序渐进，快速拿到大厂offer。**双语**教学，适合**国内外**       求职。

课程	阶段难度	面向人群
九章算法基础班 Python + Java	第一阶段（简单+中等）	CS以及相关专业出身，但对于工业界算法面试的题型和技巧不够了解的同学 非CS相关专业出身，想快速转码，但语言或算法基础薄弱的同学
九章算法班2022版 Python + Java	第二阶段（中等+难）	有一定算法基础，希望快速全面掌握算法知识体系，冲刺算法面试的同学
九章算法面试高频题冲刺班 Python + Java	第三阶段（中等+难）	最好上过算法班，已经对算法有较全面的了解和扎实的基础，希望掌握新题、高频题、难题，冲刺面试的同学

本课程（冲刺班）适用人群

- 刷题数目 > 100 题（精刷）
- 常见面试算法都接触过
- 必须精通《九章算法基础班》的知识，理解大部分《九章算法班》的知识
- **本课程不适合零基础和基础薄弱的同学**

求其上者得其中
求其中者得其下
求其下者無所得



面试算法知识点及考察频率 Cheat Sheet

数据结构知识点及考察频率 Cheat Sheet

时间复杂度与算法对应关系 Cheat Sheet

高频算法通用代码模板 Cheat Sheet

面试算法知识点及考察频率 Cheat Sheet

面试算法知识点	考察情况	学习难度	最少刷题数	哪些九章课程中讲过
字符串处理 String	考得很多，主要注重代码实现能力，算法上没有太多难点，通常是处理麻烦	低	20	九章算法基础班
排序算法 Sorting	直接考很少，一般是考察其中的快速排序和归并排序及相关的题，必须背诵这俩	低	2	九章算法基础班
双指针算法 Two Pointers	高频算法之王，变形特别多。算法不算特别难，但能快速想到和写好不容易	中	20	九章算法班、高频题冲刺班
二分法 Binary Search	考察频率中等，能写好写对不容易，二分答案的问题甚至很难想到算法，要背模板	中	10	九章算法班、高频题冲刺班
分治法 Divide & Conquer	考察频率中等，一般和二叉树一起出现和考察，题一般不难	低	10	九章算法班
宽度优先搜索 BFS	考察频率高，实现一般都不难	低	5	九章算法班
深度优先搜索 DFS / 递归 Recursion	考察频率高，主要是考递归会不会写	难	20	九章算法班、高频题冲刺班
二叉树的遍历算法 Traversal	考察频率中等、最常考中序遍历非递归	低	5	九章算法基础班、九章算法班
动态规划 Dynamic Programming	国内大厂基本都考、北美主要是 G F 喜欢考，其他公司考得很少	难	50	九章算法班、高频题冲刺班、动态规划专题班
拓扑排序算法 Topological Order	考察频率中等、但是每个公司基本都有一个这个算法的题	中	3	九章算法班

不太考的算法（别花时间）

任意图最短路算法、贪心法、带名字的各类算法，如：

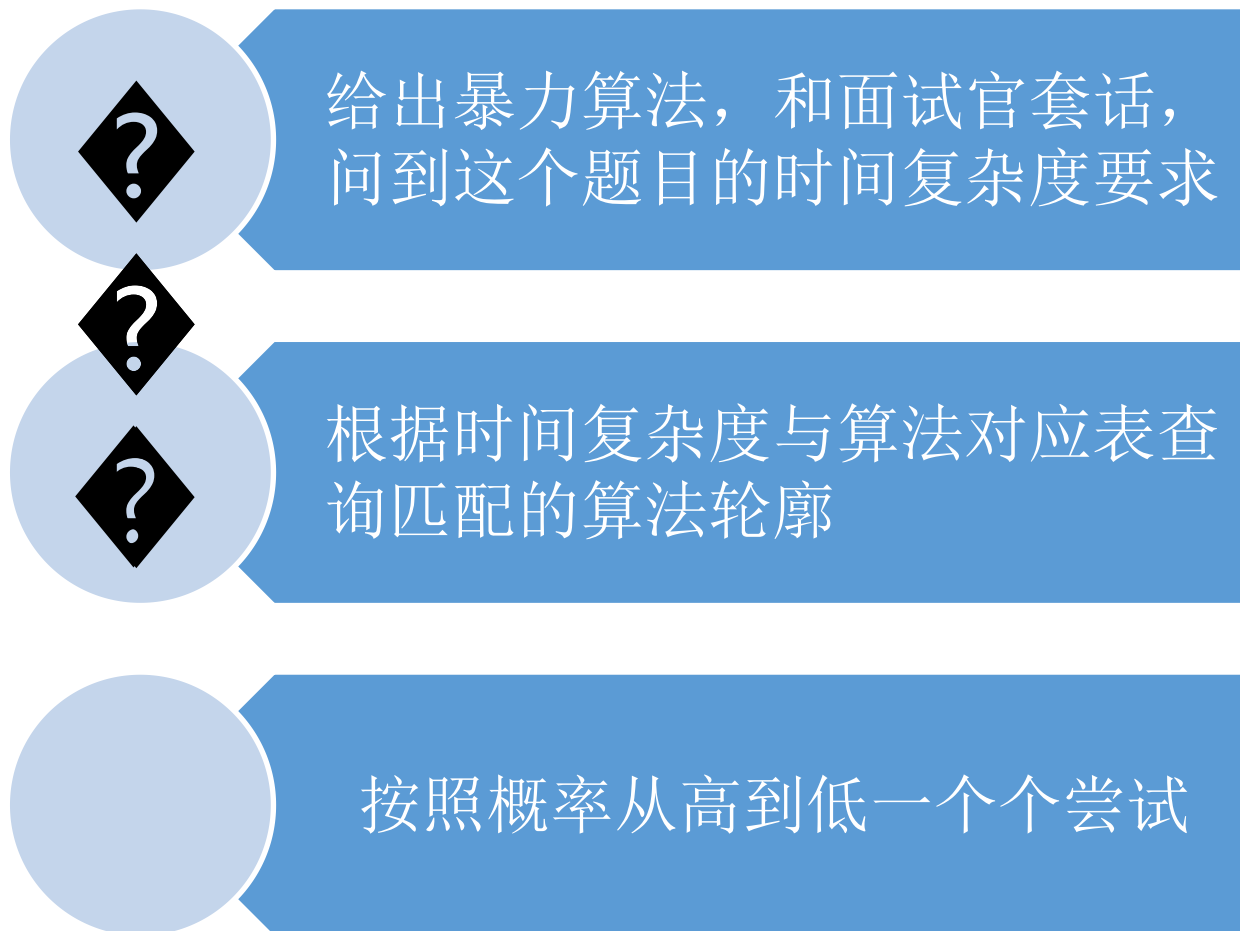
KMP, Morris, Manacher, Dinic, Dijkstra, Floyd, Prim ...

数据结构知识点及考察频率 Cheat Sheet Part 1

面试算法知识点	考察情况	学习难度	最少刷题数	哪些九章课程中讲过
链表 LinkedList	中小公司考得多，大公司近年来考得少 题目一般不难，主要考察 Reference	低	20	九章算法基础班
二叉树 Binary Search	中小公司考得多，大公司近年来考得少 题目一般不难，主要考察 Reference	低	20	九章算法基础班、九章算法班
堆 Heap	高频，经常会用到，原理必须掌握，但不用掌握代码实现。应用必须掌握代码。	中	5	九章算法班、高频题冲刺班
哈希表 Hash Table	高频，原理和应用都需要掌握且需要掌握代码实现。	中	10	九章算法基础班、九章算法班
线段树 Segment Tree	不太考，有的题目存在多种解法的时候 Segment Tree 可以帮上忙降低思考难度	中	3	线段树与树状数组
树状数组 Binary Indexed Tree	不太考，与其学这个不如学线段树	中	2	线段树与树状数组
跳跃表 Skip List	不太考，需要大致知道原理，分布式数据库里会用到这个数据结构	难	1	系统架构设计
字典树 Trie	考察频率中等，跟单词有关的问题一般多多少少都可以用到去优化，可替代哈希表	中	3	高频题冲刺班
并查集 Union Find	考察频率中等，主要是 G 和 F 可能会考，不会的话很多时候可以用 BFS 替代	中	3	高频题冲刺班
红黑树 RB-Tree	只有 G 可能会问到，也只是问大致原理，能干啥， Java 会用 TreeMap 就行	难	1	自行到 Google 去搜索

数据结构知识点及考察频率 Cheat Sheet Part 2

面试算法知识点	能干哪些事情，复杂度如何
数组 Array	$O(1)$ append, update(知道index) $O(n)$ delete(移动后面数的补空), find
链表 LinkedList	$O(1)$ insert, delete (必须知道前面的点), $O(n)$ find
二叉树 Binary Tree	最坏 $O(n)$ 增删查改, 注意二叉树的高度不是 $O(\log n)$, 是 $O(n)$ 的
堆 Heap	$O(\log n)$ push, delete, pop, $O(1)$ top
哈希表 Hash Table	$O(1)$ 增删查改, 如果 key 是字符串那就是 $O(L)$ 增删查改, L 是字符串长度
线段树 Segment Tree	$O(\log n)$ insert, delete, find, update, rangeMax, rangeMin, rangeSum, lowerBound, upperBound, $O(1)$ min, max, sum, 万能数据结构
树状数组 Binary Indexed Tree	$O(\log n)$ rangeSum
跳跃表 Skip List	$O(\log n)$ insert, delete, find, update, lowerBound, upperBound $O(1)$ max, min
红黑树 RB-Tree	$O(\log n)$ insert, delete, find, update, lowerBound, upperBound $O(1)$ max, min
字典树 Trie	$O(L)$ insert, delete, find, update, L 是字符串长度
并查集 Union Find	$O(1)$ find, union, isConnected, getSize



时间复杂度与算法对应关系 Cheat Sheet

时间复杂度	可能的算法	哪些九章课程中讲过
$O(1)$	数学题，一般不太会在面试中出现，因为面试主要考 Coding, $O(1)$ 没代码量	无
$O(\log n)$	90% 二分法 10% 倍增法（每次x2），快速幂（求 x^n ），欧几里得算法（求最大公约数）	九章算法班、高频题冲刺班
$O(\sqrt{n})$	99% 因数分解 Factorization 1% 分块检索(将n的区间分成 \sqrt{n} 个 \sqrt{n} 大小的区间，每个区间单独维护和统计数据)	九章算法班
$O(n)$	50% 双指针 Two Pointers（同向双指针、相向双指针、背向双指针、合并双指针） 20% 二叉树遍历或者分治 Binary Tree Traversal / Divide & Conquer 10% n次 $O(1)$ 的操作，每次操作一个 $O(1)$ 的数据结构 (UnionFind, HashMap) 10% 单调栈 Monotonic-stack、单调队列 Monotonic-queue 10% 枚举法 Enumeration（如打擂台算法，for 循环一个数组）	九章算法基础班、九章算法班、高频题冲刺班
$O(n \log n)$	60% n 次 $O(\log n)$ 的操作， $\log n$ 可能是二分法，也可能是操作 $\log n$ 的数据结构(堆，线段树，红黑树) 20% $\log n$ 次 $O(n)$ 的操作，一般是二分答案，用 $O(n)$ 的时间检测答案偏大偏小 20% 排序 + 其他 $O(n)$ 或者 $O(n \log n)$ 的算法	九章算法基础班、九章算法班、高频题冲刺班
$O(n \log k)$	50% n 次 $O(\log k)$ 的操作 50% 类似归并排序，分治一个 k 的区间，每一层处理时间复杂度 $O(n)$ ，共 $\log k$ 层	九章算法班
$O(n+m)$ 点+边	100% BFS	九章算法班
$O(n^2)$, $O(n^3)$...	50% $O(n)$ 枚举某个参数，降维之后用其他算法 30% 动态规划 20% 如果是 $n \times n$ 的矩阵，可能是 BFS	九章算法班、高频题冲刺班
$O(2^n)$	100% 组合相关的深度优先搜索	九章算法班
$O(n!)$	100% 排列相关的深度优先搜索	九章算法班

通过时间复杂度倒推算法举例

这个题目用什么算法？

585 Maximum Number in Mountain Sequence 山脉序列中的最大值

Given a mountain sequence of n integers which **increase firstly and then decrease**, find the mountain top(Maximum).

给 n 个整数的山脉数组，即先增后减的序列（没有相等），找到山顶（最大值）。

输入：

[1, 2, 4, 8, 6, 3]

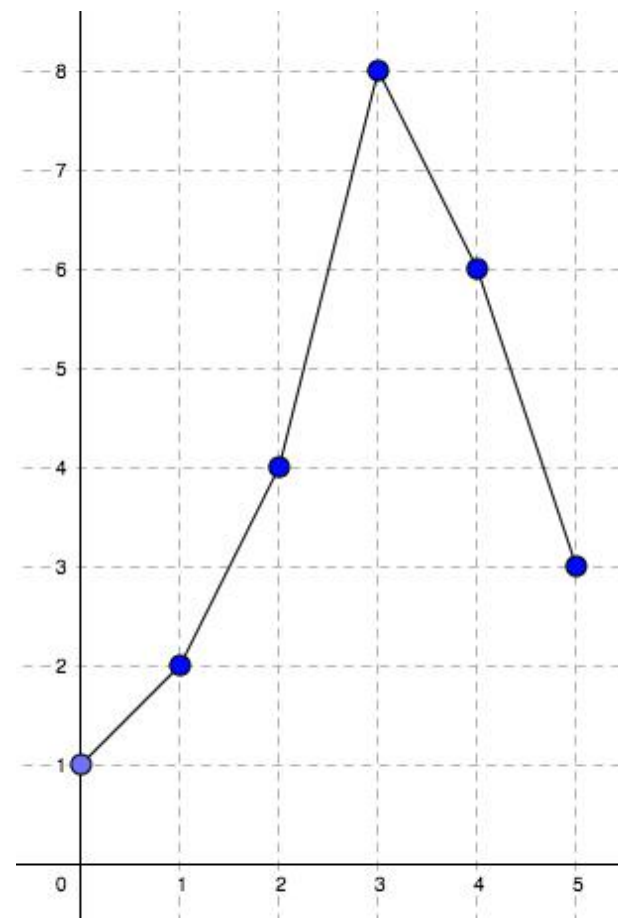
输出：

8

如果你的暴力算法是 $O(n)$ 的

面试官要求继续优化

那就要想到用 $O(\log n)$ 二分法，其他算法的出现概率都很低



高频套路：前缀和 Prefix Sum

前缀和定义

构造一个长度为 $n+1$ 的数组 `prefixSumArr`

`prefixSumArr[i]` 代表前 i 个数字的和， $\text{prefixSumArr}[i] = 0$

index	0	1	2	3	4
原数组	2	-3	4	-1	2

index	0	1	2	3	4	5
前缀和数组	0	2	-1	3	2	4

构造前缀和

当前前缀和 = 之前前缀和 + 当前元素

$\text{prefixSumArr}[i] = \text{prefixSumArr}[i - 1] + \text{arr}[i - 1]$

前缀和数组Index	0	1	2	3	4	5
原数组arr		2	-3	4	-1	2
前缀和数组 prefixSumArr	0	2	-1	3	2	4

前缀和的应用

使用前缀和数组在 $O(1)$ 的时间复杂度内计算子数组和

sum from i to $j = \text{prefixSum}[j + 1] - \text{prefixSum}[i]$

```
def get_prefix_sum(self, nums):
    prefix_sum = [0]
    for num in nums:
        prefix_sum.append(prefix_sum[-1] + num)
    return prefix_sum

private int[] getPrefixSum(int[] nums) {
    int[] prefixSum = new int[nums.length + 1];
    prefixSum[0] = 0;
    for (int i = 0; i < nums.length; i++) {
        prefixSum[i + 1] = prefixSum[i] + nums[i];
    }
    return prefixSum;
}
```

与前缀和有关的题目列表

LintCode 265. Maximum Non-Negative Subarray

LintCode 41. 42. 43. 620. 621. 722. Maximum Subarray (1~6)

LintCode 45. Maximum Subarray Difference

LintCode 1083. Maximum Sum of 3 Non-Overlapping Subarrays

LintCode 1724. Maximum Sum Circular Subarray

LintCode 868. 617. Maximum Average Subarray (1~2)

LintCode 1567. Maximum Can Exchanged Subarray

LintCode 406. Minimum Size Subarray Sum

LintCode 911. Maximum Size Subarray Sum Equals k

LintCode 191. Maximum Product Subarray

LintCode 1073. Maximum Length of Repeated Subarray

LintCode 1712. Binary Subarrays With Sum

LintCode 1258. Beautiful Subarrays

LintCode 1743. Bitwise ORs of Subarrays

LintCode 402. 403. 753. Continuous Subarray Sum (1~2)

LintCode 264. Counting Universal Subarrays

LintCode 1517. Largest subarray

让我们来牛刀小试一下

利用“时间复杂度倒推法”和“前缀和”技巧

我们来一起一步步分析和解决两道最新的大厂面试真题

1844 subarray sum equals to k II 子数组和为K II

Given an array of integers and an integer k, you need to find the minimum size of continuous **no-empty** subarrays whose sum equals to k, and return its length.

if there are no such subarray, return -1.

给定一个整数数组和一个整数k，你需要找到和为k的最短非空子数组，并返回它的长度。

如果没有这样的子数组，返回-1.

输入:

nums = [1, 1, 1, 2]

k = 3

输出:

2

解释:

[1,2]的sum = 3且长度最短

[1,1,1]的sum = 3但长度较长

输入:

nums = [2, 1, -1, 4, 2, -3]

k = 3

输出:

2

解释:

[2,1]的sum = 3且长度最短

[4,2,-3]的sum = 3但长度较长

先修知识点: subarray vs subsequence

[1, 2, 3, 4]

subarray 必须是连续的

subsequence 可以是间隔的

一个长度为 n 的数组，有多少个不同的子数组 subarray?

A: $O(n^2)$

B: $O(2^n)$

一个长度为 n 的数组，有多少个不同的子序列 subsequence?

A: $O(n^2)$

B: $O(2^n)$

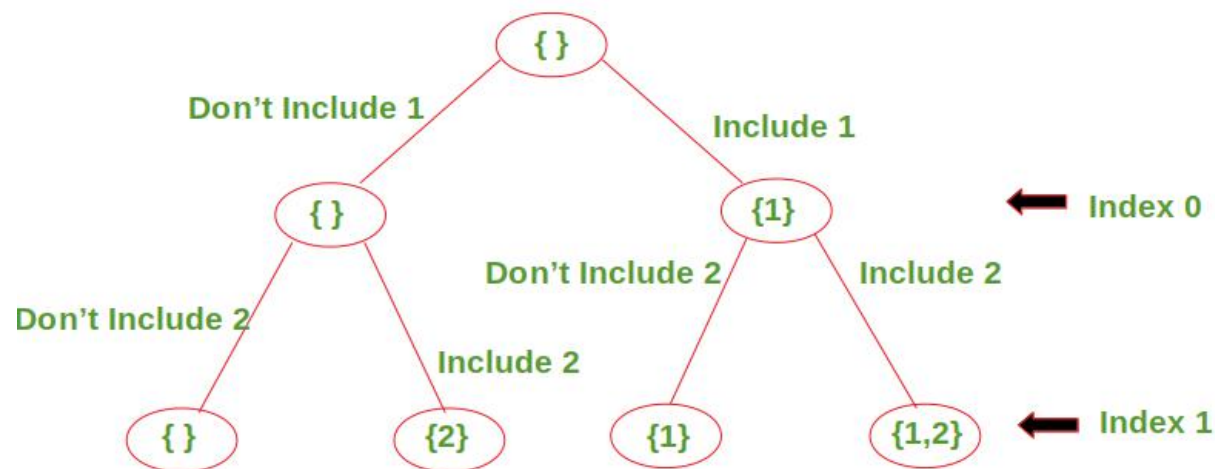
[1], [2], [3], [4]

[1,2], [2,3], [3,4],

[1,2,3], [2,3,4]

[1,2,3,4]

等比数列求和 = $(1+n) * n / 2$



题目解析 —— 暴力算法 (Brute Force)

把所有子数组都撸一遍，时间复杂度 $O(n^3)$

```
for 子数组左端点 start //  $O(n)$ 
    for 子数组右端点 end //  $O(n)$ 
        for start 到 end 求和 //  $O(n)$ 
            判断是不是 k
```

用前缀和数组在 $O(1)$ 时间内直接算得子数组和，时间复杂度 $O(n^2)$

```
for 子数组左端点 start //  $O(n)$ 
    for 子数组右端点 end //  $O(n)$ 
        判断  $\text{prefixSum}[\text{end} + 1] - \text{prefixSum}[\text{start}]$ 
            是不是 k
```

经过和面试官的深入交流

面试说：你得找一个更快的算法，希望你使用 $O(n)$ 的算法来解决



逐个尝试 $O(n)$ 的算法

查询《时间复杂度与算法对应关系 Cheat Sheet》

按照概率一个个试试，直到找到合适的算法

- ~~50% 双指针 Two Pointers (同向双指针、相向双指针、背向双指针、合并双指针)~~
- ~~20% 二叉树遍历或者分治 Binary Tree Traversal / Divide & Conquer~~
- 10% n 次 $O(1)$ 的操作，每次操作一个 $O(1)$ 的数据结构 (UnionFind, HashMap)
- 10% 单调栈 Monotonic-stack、单调队列 Monotonic-queue
- 10% 枚举法 Enumeration (如打擂台算法，for 循环一个数组)

题目解析 —— 常用套路：最大子数组

要使得 $\text{prefixSum}[j + 1] - \text{prefixSum}[i]$ 最大

对于给定 j , 找到 i from 0 to j 之间的“最小”的 $\text{prefixSum}[i]$

算法中经常会出现找最大值时要通过最小值去找

LintCode 41: Maximum Subarray

LintCode 545: Top K Largest Number II

以下标6为结尾的最大子数组之和是多少？

前缀和数组Index	0	1	2	3	4	5	6	7	8
原数组 arr		2	-3	4	-1	2	3	-7	-2
前缀和数组 prefixSumArr	0	2	-1	3	2	4	7	0	-2

如何求得以下标6为结尾的最大子数组之和？

用 dict / HashMap 记录之前的 prefixSum

K= 12 0 1 2 3 4
 [3, 1, -1, 5, 7]

PrefixSum = 15

Key (前缀和)	Value (前缀和Index)
0	0
3	3
4	2
8	4

```

7  def subarraySumEqualsKII(self, nums, k):
8      # 前缀和 -> 前缀和数组的index
9      sum_to_index_map = dict()
10     sum_to_index_map[0] = 0
11     answer = float('inf')
12     # prefix_sum是从起点到某个点的所有数字之和
13     prefix_sum = 0
14     for i in range(len(nums)):
15         # 得到从0到当前位置i的前缀和
16         prefix_sum += nums[i]
17         # 如果之前有一个前缀和为(prefix_sum - k), 满足条件
18         if (prefix_sum - k) in sum_to_index_map:
19             # 得到以i为结尾, 并且和为k的子数组的长度
20             length = i + 1 - sum_to_index_map[prefix_sum - k]
21             # 把answer更新为更小值
22             answer = min(answer, length)
23             # 存入dict, 以备后用
24             sum_to_index_map[prefix_sum] = i + 1
25     # 如果答案为正无穷, 没有找到, 返回-1; 否则, 返回答案
26     return -1 if (answer == float('inf')) else answer
    
```

```

7  public int subarraySumEqualsKII(int[] nums, int k) {
8      // 前缀和 -> 前缀和数组的index
9      HashMap<Long, Integer> sumToIndexMap = new HashMap<Long, Integer>();
10     sumToIndexMap.put((long)0, 0);
11     // prefix_sum是从起点到某个点的所有数字之和
12     long prefixSum = 0;
13     int answer = Integer.MAX_VALUE;
14     for (int i = 0; i < nums.length; i++) {
15         // 得到从0到当前位置i的前缀和
16         prefixSum = prefixSum + (long)nums[i];
17         // 如果之前有一个前缀和为(prefix_sum - k), 满足条件
18         if (sumToIndexMap.containsKey(prefixSum - k)) {
19             // 得到以i为结尾, 并且和为k的子数组的长度
20             int len = i + 1 - sumToIndexMap.get(prefixSum - k);
21             // 把answer更新为更小值
22             answer = Math.min(answer, len);
23         }
24         // 存入map, 以备后用
25         sumToIndexMap.put(prefixSum, i + 1);
26     }
27     // 如果答案为正无穷, 没有找到, 返回-1; 否则, 返回答案
28     return (answer == Integer.MAX_VALUE) ? -1 : answer;
29 }
30
    
```

Followup: 如何求和为K的最长子数组?

课间休息 不要走开，有福利！

快扶我起来
我还能学



Follow up 常见套路有哪些？

常量 \Rightarrow 变量

一维 \Rightarrow 二维

$= k \Rightarrow \geq k$

最短 \Rightarrow 最长

正数 \Rightarrow 负数

不可变数据 immutable \Rightarrow 可变数据 mutable

离线算法 offline \Rightarrow 在线算法 online, 数据流 Data Stream



我走过最长最远的路
就是你的套路

1507 Shortest Subarray with Sum at Least K 和至少为 K 的最短子数组

Return the length of the shortest, non-empty, contiguous subarray of A with sum at least K.

If there is no non-empty subarray with sum at least K, return -1.

返回 A 的**最短**的非空连续子数组的长度，该子数组的和**至少为 K**。

如果没有和至少为 K 的非空子数组，返回 -1。

输入：

A = [1]

K = 1

输出：

1

解释：

[1] 的 $\text{sum} \geq 1$ 且长度最短

输入：

A = [1, 2]

K = 4

输出：

-1

解释：

没有任何一个子数组的 $\text{sum} \geq 4$

输入：

A = [5, -1, 2, 3, -2]

K = 8

输出：

4

解释：

[5,-1,2,3] 的 $\text{sum} \geq 8$ 且长度最短

承接上题，这个题要求和 $\geq k$ 就行而不是刚好 $=k$

是否还可以用 **dict / HashMap** 呢？

不行

哈希表只适合找刚好等于某个值的数据，不适合找 \leq 或者 \geq 某个值的数据

必须熟悉数据结构 **Cheat Sheet** 中每个数据结构的功能

题目解析 —— 暴力算法 (Brute Force)

for start + for end + prefixSum 求和找到 $\geq k$ 的最短数组

时间复杂度 $O(n^2)$

经过和面试官的深入交流

面试说：你得找一个更快的算法，使用比 $O(n^2)$ 快的任意方法都行



题目解析 —— $O(n \log n)$ 的算法

查询《时间复杂度与算法对应关系表 Cheat Sheet》，按照概率把 $O(n \log n)$ 的算法一个个试过来

60% n 次 $O(\log n)$ 的操作， $O(\log n)$ 可能是二分法，也可能是操作 $O(\log n)$ 的数据结构(堆，线段树)

本题可以用线段树求解，线段树的考察相对低频

且这个题目只用线段树还不够，还需要用离散化配合

在九章领扣答案中查询线段树版本的[参考答案](#)

20% $\log n$ 次 $O(n)$ 的操作，一般是二分答案，用 $O(n)$ 的时间检测答案偏大偏小

这个题目的答案明显是有范围的！！！！

~~**20% 排序 + 其他 $O(n)$ 或者 $O(n \log n)$ 的算法**~~

数组如果排序之后会丢失 subarray 的连续性

如果是对 prefixSum 排序，试了一下之后也发现找不到思路

题目解析 —— 答案集合上的二分法

二分数组长度 x ，判断条件是哪个？

A: 检测是否存在subarray和 $\geq k$ 且长度 $= x$

B: 检测是否存在subarray和 $\geq k$ 且长度 $\leq x$

原数组	下标	0	1	2	3	4	5
	值	4	-2	3	-4	5	-6

	找到 $\text{sum} \geq 4$ 的最短子数组 答案为1						
子数组长度	0	1	2	3	4	5	6
存在subarray, $\text{sum} \geq 4$ && 长度 $= x$	✗	✓	✗	✓	✗	✓	✗
存在subarray, $\text{sum} \geq 4$ && 长度 $\leq x$	✗	✓	✓	✓	✓	✓	✓

4,

找到 $\text{sum} \geq 8$ 的最短子数组

下标	0	1	2	3	4
原数组	5	-1	2	3	-2

		下标	0	1	2	3	4	5
		前缀和	0	5	4	6	9	7

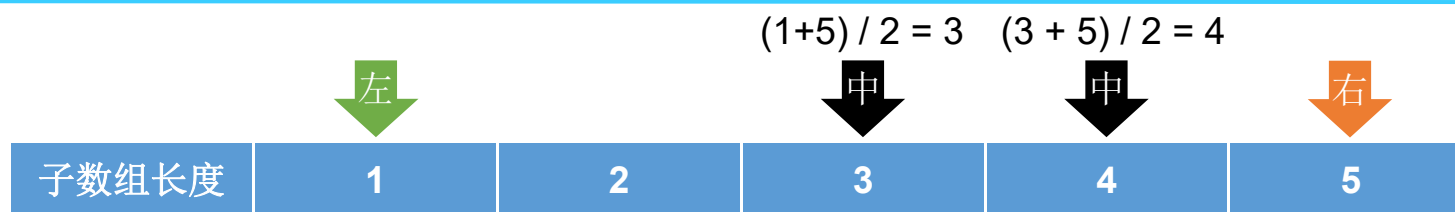
			$(1 + 5) / 2 = 3$	$(3 + 5) / 2 = 4$	
	左		中	中	右
子数组长度	1	2	3	4	5

判断是否存在一个子数组，满足：(子数组之和 $\geq K$) and (子数组长度 $\leq \text{length}$)

代码解析 —— 答案集合上的二分法

[5, -1, 2, 3, -2]

找到 $\text{sum} \geq 8$ 的最短子数组



```

4  def shortestSubarray(self, A, K):
5      # 构建前缀和
6      prefix_sum = self.get_prefix_sum(A)
7
8      # 在答案集合上进行二分, 答案最短为1个数字, 答案最长为全部数字
9      start, end = 1, len(A)
10
11     # 九章算法经典二分模板
12     while start + 1 < end:
13         mid = (start + end) // 2
14         if self.is_valid(prefix_sum, K, mid):
15             end = mid
16         else:
17             start = mid
18
19     # 寻找最短子数组, 所以先验证start, 再验证end
20     if self.is_valid(prefix_sum, K, start):
21         return start
22     if self.is_valid(prefix_sum, K, end):
23         return end
24
25     # 无解, 返回-1
26     return -1
27
28     # 构建前缀和
29     def get_prefix_sum(self, A):
30         prefix_sum = [0]
31         for num in A:
32             prefix_sum.append(prefix_sum[-1] + num)
33         return prefix_sum
    
```

```

2  public int shortestSubarray(int[] A, int K) {
3      // 构建前缀和
4      int[] prefixSum = getPrefixSum(A);
5      // 在答案集合上进行二分, 答案最短为1个数字, 答案最长为全部数字
6      int start = 1, end = A.length;
7      // 九章算法经典二分模板
8      while (start + 1 < end) {
9          int mid = (start + end) / 2;
10         if (isValid(prefixSum, K, mid)) {
11             end = mid;
12         } else {
13             start = mid;
14         }
15     }
16     // 寻找最短子数组, 所以先验证start, 再验证end
17     if (isValid(prefixSum, K, start)) {
18         return start;
19     }
20     if (isValid(prefixSum, K, end)) {
21         return end;
22     }
23     // 无解, 返回-1
24     return -1;
25 }
26
27 // 构建前缀和
28 private int[] getPrefixSum(int[] A) {
29     int[] prefixSum = new int[A.length + 1];
30     prefixSum[0] = 0;
31     for (int i = 0; i < A.length; i++) {
32         prefixSum[i + 1] = prefixSum[i] + A[i];
33     }
34     return prefixSum;
35 }
    
```

	下标	0	1	2	3	4	5
	前缀和	0	5	4	6	9	7

红框需要实现哪些操作？

- 红框右移后快速加入最右元素
- 红框右移后快速删除最左元素（这里删除的元素不一定是堆顶元素，不能直接 `pop` 堆顶）
- 在红框内的 `x` 个 `prefixSum` 中求最小值

红框用什么数据结构实现？

- `Heap (PriorityQueue)` 原生只支持 `O(N)` 删除，需要自己实现 `log(N)` 删除给定元素
- `Segment Tree` 需要自己实现 `Segment Tree`
- `RB Tree` Java 中的 `TreeMap`，Python 原生不支持

	下标	0	1	2	3	4	5
	前缀和	0	5	4	6	9	7

```

35 def is_valid(self, prefix_sum, K, length):
36     # 创建最小堆
37     minheap = Heap()
38     for end in range(len(prefix_sum)):
39         # 淘汰之前元素
40         index = end - length - 1
41         # 初始阶段, 只有入堆没有出堆, 如果index小于0, 无需删除
42         if index >= 0:
43             minheap.delete(index)
44         # 找到一个长度 <= length 且 sum >= K 的子数组, 立即返回true
45         if not minheap.is_empty() and \
46             prefix_sum[end] - minheap.top()[0] >= K:
47             return True
48         # 加入新元素
49         minheap.push(end, prefix_sum[end])
50     # 无法找到满足条件的子数组, 返回false
51     return False
    
```

```

37 // 判断是否存在一个子数组, 满足: (子数组之和 >= K) and (子数组长度 <= length)
38 private boolean isValid(int[] prefixSum, int K, int length) {
39     // 创建最小堆
40     Heap minheap = new Heap();
41
42     for (int end = 0; end < prefixSum.length; end++) {
43         // 淘汰之前元素
44         int index = end - length - 1;
45
46         // 初始阶段, 只有入堆没有出堆, 如果index小于0, 无需删除
47         if (index >= 0) {
48             minheap.delete(index);
49         }
50
51         // 找到一个长度 <= length 且 sum >= K 的子数组, 立即返回true
52         if (!minheap.isEmpty() && prefixSum[end] - minheap.top().val >= K) {
53             return true;
54         }
55         // 加入新元素
56         minheap.push(end, prefixSum[end]);
57     }
58     // 无法找到满足条件的子数组, 返回false
59     return false;
60 }
    
```

```
53 # 可以log(n)删除的heap
54 class Heap:
55     def __init__(self):
56         self.minheap = []
57         # 用于标记被删除的元素 (软删除)
58         self.deleted_set = set()
59
60     # 把元素加入最小堆
61     def push(self, index, val):
62         # Python自带的heappush
63         heappush(self.minheap, (val, index))
64
65     # 如果栈顶元素已经被标记为删除, 则把该元素从栈顶移除
66     def _lazy_deletion(self):
67         # 如果栈顶元素已经被标记为删除
68         while self.minheap and self.minheap[0][1] in self.deleted_set:
69             # 移除栈顶的被删除元素
70             prefix_sum, index = heappop(self.minheap)
71             # 本题没有这行代码也可以成功通过测试用例, 但是把历史数据清理干净是好习惯
72             self.deleted_set.remove(index)
73
74     # peek栈顶元素 (不移除)
75     def top(self):
76         self._lazy_deletion()
77         return self.minheap[0]
78
79     # pop栈顶元素 (移除)
80     def pop(self):
81         self._lazy_deletion()
82         heappop(self.minheap)
83
84     # 标记一个元素被删除 (并没有真正从栈中移除)
85     def delete(self, index):
86         self.deleted_set.add(index)
87
88     # 堆是否为空
89     def is_empty(self):
90         return not bool(self.minheap)
```

```
72 // 可以log(n)删除的heap
73 class Heap {
74     // Java自带的PriorityQueue(heap)
75     private Queue<ValueIndexPair> minheap;
76     // 用于标记被删除的元素 (软删除)
77     private Set<Integer> deleteSet;
78
79     public Heap() {
80         minheap = new PriorityQueue<>((p1, p2) -> (p1.val - p2.val));
81         deleteSet = new HashSet<>();
82     }
83
84     // 把元素加入最小堆
85     public void push(int index, int val) {
86         minheap.add(new ValueIndexPair(val, index));
87     }
88
89     // 如果栈顶元素已经被标记为删除, 则把该元素从栈顶移除
90     private void lazyDeletion() {
91         // 如果栈顶元素已经被标记为删除
92         while (minheap.size() != 0 &&
93             deleteSet.contains(minheap.peek().index)) {
94             // 移除栈顶的被删除元素
95             ValueIndexPair pair = minheap.poll();
96             // 已经被移除, 从deleteSet中删掉
97             // 本题没有这行代码也可以成功通过测试用例, 但是把历史数据清理干净是好习惯
98             deleteSet.remove(pair.index);
99         }
100     }
101
102     // peek栈顶元素 (不移除)
103     public ValueIndexPair top() {
104         lazyDeletion();
105         return minheap.peek();
106     }
107
108     // pop栈顶元素 (移除)
109     public void pop() {
110         lazyDeletion();
111         minheap.poll();
112     }
113
114     // 标记一个元素被删除 (并没有真正从栈中移除)
115     public void delete(int index) {
116         deleteSet.add(index);
117     }
118
119     // 堆是否为空
120     public boolean isEmpty() {
121         return minheap.isEmpty();
122     }
123 }
```

时间复杂度分析 —— 支持 $O(\log n)$ 删除任意元素的堆

用懒惰删除算法支持删除给定元素的 **Heap** 的时间复杂度分别为：

操作	时间复杂度
push, pop, top	$O(\log n)$
delete	$O(1)$

为什么 **top** 和 **pop** 里有 **while** 循环依然是 $O(\log n)$ 呢？

while 循环不会每次都是最坏情况，需要将总体耗费均摊到每个元素上的处理上（不懂没关系，记住就好）

算法步骤回顾

1. $\log n$ 二分 subarray 的长度 x
2. for subarray 右端点 end
3. 调整堆里的元素, 求得最小值 min , 判断 $prefixSum[end + 1] - min$ 是不是 $\geq k$

时间复杂度 $O(\log n \text{ 二分} * n \text{ 枚举 } end * \log n \text{ 堆操作}) = O(n * (\log n)^2) > O(n^2)$

虽然不如 $O(n)$, $O(n \log n)$ 好, 但也是可以被面试官接受的

题目解析 & 代码解析 —— 最优的 $O(n)$ 算法，单调队列

本问题中的 $O(n)$ 的方法需要使用单调队列(monotonic queue)来解决，这个内容我们将在第九章直播课中讲解

```
1 from collections import deque
2
3 class Solution:
4     def shortestSubarray(self, A, k):
5         # 双端队列，存放前缀和数组的下标
6         # 注意：存了下标，就不需要存放值
7         deq = deque([0])
8
9         # 前缀和数组
10        prefix_sum = self.get_prefix_sum(A)
11        min_length = float('inf')
12
13        for i in range(len(prefix_sum)):
14            # 如果前i个数的前缀和 - deque左端的前缀和 >= K,
15            # 则是满足条件的可行解。用可行解的长度更新最小长度
16            while deq and prefix_sum[i] - prefix_sum[deq[0]] >= k:
17                min_length = min(min_length, i - deq.popleft())
18
19            # 如果prefixSum[i] <= prefixSum[deque.getLast()], 那么
20            # 就需要淘汰prefixSum[deque.getLast()]。prefixSum[i]是比
21            # prefixSum[deque.getLast()]更优秀的区间起点，因为prefixSum[i]
22            # 的位置更靠后（题目要求长度最短），且相减之后区间和会更小（或相等）
23            while deq and prefix_sum[i] <= prefix_sum[deq[-1]]:
24                deq.pop()
25
26            # i可能是后面会用到的一个区间起点，入deque。单调队列每个元素都会入队
27            deq.append(i)
28
29        return -1 if min_length == float("inf") else min_length
30
31        # 构建前缀和数组
32        def get_prefix_sum(self, A):
33            prefix_sum = [0]
34            for n in A:
35                prefix_sum.append(prefix_sum[-1] + n)
36            return prefix_sum
```

```
1 public class Solution {
2     /**
3      * @param A: the array
4      * @param K: sum
5      * @return: the length
6      */
7     public int shortestSubarray(int[] A, int K) {
8         // 双端队列，存放前缀和数组的下标
9         // 注意：存了下标，就不需要存放值
10        Deque<Integer> deque = new ArrayDeque<>();
11        // 前缀和数组
12        int[] prefixSum = getPrefixSum(A);
13        int minLength = Integer.MAX_VALUE;
14
15        for (int i = 0; i < prefixSum.length; i++) {
16            // 如果前i个数的前缀和 - deque左端的前缀和 >= K,
17            // 则是满足条件的可行解。用可行解的长度更新最小长度
18            while (!deque.isEmpty() &&
19                prefixSum[i] - prefixSum[deque.getFirst()] >= K) {
20                minLength = Math.min(minLength, i - deque.pollFirst());
21            }
22            // 如果prefixSum[i] <= prefixSum[deque.getLast()], 那么
23            // 就需要淘汰prefixSum[deque.getLast()]。prefixSum[i]是比
24            // prefixSum[deque.getLast()]更优秀的区间起点，因为prefixSum[i]
25            // 的位置更靠后（题目要求长度最短），且相减之后区间和会更小（或相等）
26            while (!deque.isEmpty() &&
27                prefixSum[i] <= prefixSum[deque.getLast()]) {
28                deque.pollLast();
29            }
30            // i可能是后面会用到的一个区间起点，入deque。单调队列每个元素都会入队
31            deque.offer(i);
32        }
33
34        return minLength == Integer.MAX_VALUE ? -1 : minLength;
35    }
36
37    // 构建前缀和数组
38    private int[] getPrefixSum(int[] A) {
39        int[] prefixSum = new int[A.length + 1];
40        for (int i = 1; i <= A.length; i++) {
41            prefixSum[i] = prefixSum[i - 1] + A[i - 1];
42        }
43        return prefixSum;
44    }
45 }
```


一个题可以有多种实现方法，都需要掌握么？

算法	时间复杂度	思维复杂度	实现复杂度
暴力枚举算法	$O(n^2)$	低	低
二分答案 + 堆	$O(n(\log n)^2)$	中	中
线段树 + 离散化	$O(n \log n)$	中下	高
单调队列	$O(n)$	高	低

各种算法各有优劣，在平时备战的时候，尽可能的都掌握

熟悉各类算法和数据结构的特性

不要追求把题做多，要追求把题做深

总结了如下一些算法和数据结构的代码模板、使用场景、典型例题

所有模板都具备非常高的通用性

BFS, DFS, Topological Sorting, BST Iterator

Binary Search, Two Pointers 四种类型, Divide & Conquer

Heap(with deletion), Segment Tree, Trie, Union Find

Dynamic Programming 3 种常考类型

请同学们课下找班主任领取

其他用到需要删除给定元素操作的题

LintCode 131: The Skyline Problem (Building Outline)

LintCode 859: Max Stack

每期更新 15% 新题，难题，高频题

10 节直播课 (每节2小时) + 20 节互动课 (每节1小时) = 40课时

1. 算法面试高频知识点与技巧
2. 面试中的高频算法：同向双指针
3. BFS进阶与最短路算法
4. DFS进阶
5. 动态规划进阶（上）
6. 动态规划进阶（下）
7. 连通性问题通解：并查集
8. 字典树的常见考点及题目解析
9. 不易想到的 $O(n)$ 算法：单调栈
10. 堆的解题技巧

Q & A