Data Structure and Algorithms Leetcode Coding Review

Jiaxin Zhang

2022 年 5 月 26 日

目录

1 Python basic knowledge

2 Double pointer and sliding window

```
# Double pointer and sliding window

## 使用条件
1. 滑动窗口 - 90% 的概率
2. 时间复杂度要求 O(n) - 80%的概率
3. 要求原地操作，只可以交换使用，不能使用额外空间，所以空间复杂度 O(1) - 80%
4. 有子数组 subarray，子字符串 substring 的关键词 - 50%
5. 有回文问题 palindrome 关键词 - 50%

## time complexity
>> 时间复杂度与最内层循环主体的 loop 执行次数有关，与有多少重循环无关，O(n)
## space complexity
>> 只需要分配 2 个指针的额外内存，所以 space 是 O(1)

## 几种类型的双指针及相关题目
1. 同向：特点是指针不回头，全 0 子串数量 - slow，fast，基本等价于 sliding window
2. 相向：two sum， three sum， left， right
3. 背向：最长回文子串

## template
### 相向双指针 - patition in quicksort
```python
class Solution:
    def patition(self, A, start, end):
        if start >= end:
            return
        left, right = start, end
        # key point 1: pivot is the value, not the index
        pivot = A[(start + end) // 2]
        # key point 2: every time you compare left & right, it should be left <= right not left < right
        while left <= right:
            while left <= right and A[left] < pivot:
                left += 1
            while left <= right and A[right] > pivot:
                right -= 1
            if left <= right:
                A[left], A[right] = A[right], A[left]
                left += 1
                right -= 1
```
背向双指针
```python
```
```

```python
class Solution:
    def patition(self, A, start, end):
        left = position
        right = position + 1
        while left >=0 and right < len(s):
            if left and right 可以停下来了:
                break
            left -= 1
            right += 1
```
同向双指针 - 快慢指针

```python
class Solution:
    def patition(self, A, start, end):
        j = 0
        for i in range(n):
            # 不满足则循环到满足搭配为止
            while j < n and i and j 之间不满足条件:
                j += 1
            if i 到 j 之间满足条件:
                处理 i 到 j 这段区间
```

高频题目整理

相向双指针
- *#1. Two Sum*
    > 暴力 double for loop -> hashtable -> 排序双指针 (如何排序 + index 操作需要注意); 这里要求返回下面，如果返回值比较容易
- *#167. Two Sum II - Input Array Is Sorted https://leetcode.com/problems/two-sum-ii-input-array-is-sorted/*
    > 排序之后返回 index，比 two sum 简单，我觉得只有排序之后，才能相向双指针，否则没有意义
- *#15. 3Sum https://leetcode.com/problems/3sum/*
    > 最外层 for loop 作为一个指针，内嵌 while loop，考虑 left，right 指针，这个难点在于去重，去重很多种办法，包括 set，还有左右移动，因为已经排序，所以用相邻位置的比较来去重
- *#16. 3Sum Closest https://leetcode.com/problems/3sum-closest/*
    > 比 15 简单，几个 edge cases 都考虑到了
- *#259. 3Sum Smaller https://leetcode.com/problems/3sum-smaller/*
    > res += right - left # 这步是关键，之前没有想清楚，为什么是 *right-left*，其实就是中间的都可以
- *#18. 4Sum https://leetcode.com/problems/4sum/*
    > 完全和 3sum 一样! 就是复杂的一点，time o(n^3)，space o(n)
- *#454. 4Sum II https://leetcode.com/problems/4sum-ii/*
    > hashmap，time o(n^2)，space o(n^2)
- *#75. Sort Colors https://leetcode.com/problems/sort-colors/*
- *#1229. Meeting Scheduler https://leetcode.com/problems/meeting-scheduler*
- *#125. Valid Palindrome https://leetcode.com/problems/valid-palindrome*

背向双指针
- *#5. Longest Palindromic Substring https://leetcode.com/problems/longest-palindromic-substring*
- *#408. Valid Word Abbreviation https://leetcode.com/problems/valid-word-abbreviation*
- *#409. Longest Palindrome https://leetcode.com/problems/longest-palindrome*
- *#680. Valid Palindrome II https://leetcode.com/problems/valid-palindrome-ii*

Sliding windows
- #3. Longest Substring Without Repeating Characters https://leetcode.com/problems/longest-substring-without-repeating-characters
- #76. Minimum Window Substring https://leetcode.com/problems/minimum-window-substring
- #1004. Max Consecutive Ones III https://leetcode.com/problems/max-consecutive-ones-iii
- #209. Minimum Size Subarray Sum https://leetcode.com/problems/minimum-size-subarray-sum
- #1438. Longest Continuous Subarray With Absolute Diff Less Than or Equal to Limit https://leetcode.com/problems/longest-continuous-subarray-with-absolute-diff-less-than-or-equal-to-limit

新添加一些题
- #38. Count and Say https://leetcode.com/problems/count-and-say (用到拼接的思想，如何双指针)
- #30. Substring with Concatenation of All Words https://leetcode.com/problems/substring-with-concatenation-of-all-words (sliding window 好题)
- #228. Summary Ranges https://leetcode.com/problems/summary-ranges/

题目答案和分析

1. Two Sum
```python
# double pointer
class Solution:
    def twoSum(self, nums, target):
        nums = [(number, index) for index, number in enumerate(nums)]
        nums.sort()
        left, right = 0, len(nums) - 1
        while left < right:
            if nums[left][0] + nums[right][0] < target: # 小于 target，left 右移
                left += 1
            elif nums[left][0] + nums[right][0] > target: # 大于 target，right 左移
                right -= 1
            else: # 这个必须要有，就是说 == target，直接返回
                return sorted([nums[left][1], nums[right][1]]) # 是否拍需要看要求，这个需要从小到大，就排一下
        return

# hashmap solution - better one!
class Solution:
    def twoSum(self, nums, target):
        if not nums: return
        n = len(nums)
        hashmap = {}
        for i in range(n):
            residual = target - nums[i]
            if residual in hashmap:
                res = [hashmap[residual], i]
            hashmap[nums[i]] = i
        return sorted(res)
```

1099. Two Sum Less Than K https://leetcode.com/problems/two-sum-less-than-k/

Given an array nums of integers and integer k, return the maximum sum such that there exists i < j with nums[i] + nums[j] = sum and sum < k. If no i, j exist satisfying this equation, return -1.
```
Example 1:

Input: nums = [34,23,1,24,75,33,54,8], k = 60
Output: 58
Explanation: We can use 34 and 24 to sum 58 which is less than 60.
Example 2:

Input: nums = [10,20,30], k = 15
Output: -1
Explanation: In this case it is not possible to get a pair sum less that 15.
```
> two sum 的一个变种!
```python
class Solution:
    def twoSumLessThanK(self, nums: List[int], k: int) -> int:
        # brute force 不对, 草!     time o(n^2) and space o(1)
        res = -1 # 这个最开始就应该等于-1
        for i in range(len(nums)):
            for j in range(i + 1, len(nums)): # 不一样的数
                twosum = nums[i] + nums[j]
                if twosum < k:
                    res = max(res, twosum)
        return res
        # double pointer - 这个思路就很清晰, time o(nlogn), space o(logn) to o(n)
        nums.sort()
        left, right = 0, len(nums) - 1
        res = - 1
        while left < right:
            twosum = nums[left] + nums[right]
            if twosum < k:
                left += 1
                res = max(res, twosum)
            else:
                right -= 1
        return res
```

167. Two Sum II - Input Array Is Sorted https://leetcode.com/problems/two-sum-ii-input-array-is-sorted/
Given a 1-indexed array of integers numbers that is already sorted in non-decreasing order, find two numbers such that they add up to a specific target number. Let these two numbers be numbers[index1] and numbers[index2] where 1 <= index1 < index2 <= numbers.length. Return the indices of the two numbers, index1 and index2, added by one as an integer array [index1, index2] of length 2. The tests are generated such that there is exactly one solution. You may not use the same element twice. Your solution must use only constant extra space.
```
Example 1:

Input: numbers = [2,7,11,15], target = 9
Output: [1,2]
```

```
Explanation: The sum of 2 and 7 is 9. Therefore, index1 = 1, index2 = 2. We return [1, 2].
Example 2:

Input: numbers = [2,3,4], target = 6
Output: [1,3]
Explanation: The sum of 2 and 4 is 6. Therefore index1 = 1, index2 = 3. We return [1, 3].
Example 3:

Input: numbers = [-1,0], target = -1
Output: [1,2]
Explanation: The sum of -1 and 0 is -1. Therefore index1 = 1, index2 = 2. We return [1, 2].
```
> two sum 变种
```python
class Solution:
    def twoSum(self, numbers: List[int], target: int) -> List[int]:
        # 就是 two sum with double pointer but it is sorted
        # 要求 o(1) space
        left, right = 0, len(numbers) - 1
        while left < right:
            if numbers[left] + numbers[right] < target:
                left += 1
            elif numbers[left] + numbers[right] > target:
                right -= 1
            else:
                return [left + 1, right + 1]
        return []
```

15. 3Sum https://leetcode.com/problems/3sum/
```python
class Solution:
    def threeSum(self, nums):
        if len(nums) < 3:
            return

        n = len(nums)
        nums.sort()
        ans = []
        for i in range(n):
            left = i + 1
            right = n - 1
            if nums[i] > 0: # 最小值大于 0
                break
            if i >= 1 and nums[i] == nums[i-1]: # 差一点，i >=1 才行，这种去重的题还是挺不好弄的!
                continue # 如果相邻的重复，那么就 move 一下，去掉重复的，直接跳过这个 loop
            # left, right = 0, n - 1 # 不是这样的，要和 i 相关
            while left < right:
                target = nums[i] + nums[left] + nums[right]
                if target < 0:
                    left += 1
                elif target > 0:
```

```
                    right -= 1
                else:
                    ans.append([nums[i], nums[left], nums[right]])
                    while left != right and nums[left] == nums[left + 1]: # 去重 left
                        left += 1
                    while left != right and nums[right] == nums[right - 1]: # 去重 right
                        right -= 1
                    left += 1
                    right -= 1
        return ans
```

18. 4Sum https://leetcode.com/problems/4sum/
```python
# hashmap solution
class Solution:
    def fourSum(self, nums: List[int], target: int) -> List[List[int]]:
        hashmap = {}
        for i in nums1:
            for j in nums2:
                if i + j in hashmap:
                    hashmap[i + j] += 1
                else:
                    hashmap[i + j] = 1
        res = 0
        for m in nums3:
            for n in nums4:
                if 0 - m - n in hashmap:
                    res += hashmap[0 - m - n]
        return res

# double pointer solution
class Solution:
    def fourSum(self, nums: List[int], target: int) -> List[List[int]]:
        # 完全和 3sum 一样! 就是复杂的一点, time o(n^3), space o(n)
        nums.sort()
        n = len(nums)
        res = []
        for i in range(n):
            if i > 0 and nums[i] == nums[i - 1]: continue
            for j in range(i+1, n):
                if j > i + 1 and nums[j] == nums[j - 1]: continue
                left = j + 1
                right = n - 1
                while left < right:
                    sum_all = nums[i] + nums[j] + nums[left] + nums[right]
                    if sum_all < target:
                        left += 1
                    elif sum_all > target:
                        right -= 1
                    else:
                        res.append([nums[i], nums[j], nums[left], nums[right]])
```

```
                        while left < right and nums[left] == nums[left + 1]:
                            left += 1
                        while left < right and nums[right] == nums[right - 1]:
                            right -= 1

                        left += 1
                        right -= 1

        return res # 位置错了，在最外层，容易忽视!
```


187. Repeated DNA Sequences https://leetcode.com/problems/repeated-dna-sequences/

The DNA sequence is composed of a series of nucleotides abbreviated as 'A', 'C', 'G', and 'T'.

For example, "ACGAATTCCG" is a DNA sequence.
When studying DNA, it is useful to identify repeated sequences within the DNA.

Given a string s that represents a DNA sequence, return all the 10-letter-long sequences (substrings) that occur more than once in a DNA molecule. You may return the answer in any order.

Example 1:
```
Input: s = "AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT"
Output: ["AAAAACCCCC","CCCCCAAAAA"]
```
Example 2:
```
Input: s = "AAAAAAAAAAAAA"
Output: ["AAAAAAAAAA"]
```
```python
# 不难，应该不算典型的 sliding window!
class Solution:
    def findRepeatedDnaSequences(self, s: str) -> List[str]:
        if not s: return
        n = len(s)
        hashm = {}
        res = []
        for i in range(n - 10 + 1): # 别忘了 +1
            curr = s[i:i + 10]
            hashm[curr] = hashm.get(curr, 0) + 1
            if hashm[curr] > 1:
                res.append(curr)

        return set(res) # 这个要加 set 否则输出一样的
        # time o(n), space o(n)
```

209. Minimum Size Subarray Sum https://leetcode.com/problems/minimum-size-subarray-sum/

Given an array of positive integers nums and a positive integer target, return the minimal length of a contiguous subarray [numsl, numsl+1, ..., numsr-1, numsr] of which the sum is greater than or equal to target. If there is no such subarray, return 0 instead.

Example 1:

```
Input: target = 7, nums = [2,3,1,2,4,3]
Output: 2
Explanation: The subarray [4,3] has the minimal length under the problem constraint.
```

Example 2:

```
Input: target = 4, nums = [1,4,4]
Output: 1
```

Example 3:

```
Input: target = 11, nums = [1,1,1,1,1,1,1,1]
Output: 0
```

```python
# sliding window 模板题，不难，涉及 subarray
class Solution:
    def minSubArrayLen(self, target: int, nums: List[int]) -> int:
        if sum(nums) < target: # 临界状态
           return 0
        n = len(nums)
        slow = 0
        res = inf
        sum_ = 0
        for fast in range(n):
            sum_ += nums[fast]
            while sum_ >= target:
                res = min(res, fast - slow + 1)
                sum_ -= nums[slow]
                slow += 1
        return res
```

□□□□ 217, 219, 220 是连续三个 contains duplicate 比较常见

217. Contains Duplicate https://leetcode.com/problems/contains-duplicate/

Given an integer array nums, return true if any value appears at least twice in the array, and return false if every element is distinct.

```
Example 1:
Input: nums = [1,2,3,1]
Output: true
Example 2:
Input: nums = [1,2,3,4]
Output: false
```

Example 3:
Input: nums = [1,1,1,3,3,4,3,2,4,2]
Output: true
```

```python
class Solution:
    def containsDuplicate(self, nums: List[int]) -> bool:
        # way 1 - hashtable
        record = set()
        for num in nums:
            if num not in record: # 注意是 not in 不是 is not in
                record.add(num)
            else:
                return True
        return False
        # time o(n), space o(n)

        # way 2 - sorting - optimal space - time o(nlogn), space o(1)
        nums.sort()
        print(nums)
        n = len(nums)
        for i in range(1, n):
            if nums[i-1] == nums[i]: # 要搞清楚题目含义
                return True
        return False
```

219. Contains Duplicate II https://leetcode.com/problems/contains-duplicate-ii/
Given an integer array nums and an integer k, return true if there are two distinct indices i and j in the array such that nums[i] == nums[j] and abs(i - j) <= k.

Example 1:
```
Input: nums = [1,2,3,1], k = 3
Output: true
```
Example 2:
```
Input: nums = [1,0,1,1], k = 1
Output: true
```
Example 3:
```
Input: nums = [1,2,3,1,2,3], k = 2
Output: false
```

```python
class Solution:
    def containsNearbyDuplicate(self, nums: List[int], k: int) -> bool:
```
```

```
        # brute force - 超时
        for i in range(len(nums)):
            for j in range(i + 1, len(nums)):
                if nums[i] == nums[j] and abs(i-j) <= k:
                    return True
        return False

        # hashtable 还是做出来了，之前方向不对!
        # 思路是，用 hash 找到所有相同的数，然后要注意更新，因为是一次遍历，如果之前的相等的数不 work 要 update 到当前值
        hashm = {}
        for i in range(len(nums)):
            if nums[i] not in hashm:
                hashm[nums[i]] = i
            else:
                dis = abs(i - hashm[nums[i]])
                if dis <= k:
                    return True
                else:
                    hashm[nums[i]] = i # 这步是关键，之前没有更新 nums = [1,0,1,1], k = 1 就过不去
                    continue
        return False
        time o(n), space o(n)

        # sliding window, 但不容易，容易顺序错了!
        s = set()
        for i, num in enumerate(nums):
            # 先判断是不是有违法的窗口
            if i > k:
                s.remove(nums[i - k - 1])
            # 如果没有违法的，看是否满足目标
            if num in s:
                return True
            # 上面都没有满足，加入 set 来判断，这三条顺序都不能变!
            s.add(num)
        return False
```

220. Contains Duplicate III https://leetcode.com/problems/contains-duplicate-iii/

Given an integer array nums and two integers k and t, return true if there are two distinct indices i and j in the array such that abs(nums[i] - nums[j]) <= t and abs(i - j) <= k.

```
Example 1:
Input: nums = [1,2,3,1], k = 3, t = 0
Output: true
Example 2:
Input: nums = [1,0,1,1], k = 1, t = 2
Output: true
Example 3:
Input: nums = [1,5,9,1,5,9], k = 2, t = 3
Output: false
```

```
> 挺难的，不太了解，超出了通常 sliding window 的范围！在 219 的基础上可能做出来一些，但排序已经 bucket 不好做

``` python
# brute force 过不了，意义不大，太 easy 了
# class Solution:
#     def containsNearbyAlmostDuplicate(self, nums: List[int], k: int, t: int) -> bool:
        # # brute force  - o(n^2), space o (1)
        # n = len(nums)
        # for i in range(n):
        #     for j in range(i + 1, n):
        #         if abs(nums[i] - nums[j]) <= t and abs(i - j) <= k:
        #             return True
        # return False
class Solution:
    def containsNearbyAlmostDuplicate(self, nums: List[int], k: int, t: int) -> bool:
        from sortedcontainers import SortedSet
        if not nums or t < 0: return False      # Handle special cases
        ss, n = SortedSet(), 0                  # Create SortedSet. `n` is the size of sortedset, max value of
        ↪  `n` is `k` from input
        for i, num in enumerate(nums):
            ceiling_idx = ss.bisect_left(num)  # index whose value is greater than or equal to `num`
            floor_idx = ceiling_idx - 1        # index whose value is smaller than `num`
            if ceiling_idx < n and abs(ss[ceiling_idx]-num) <= t: return True  # check right neighbour
            if 0 <= floor_idx and abs(ss[floor_idx]-num) <= t: return True     # check left neighbour
            ss.add(num)
            n += 1
            if i - k >= 0:  # maintain the size of sortedset by finding & removing the earliest number in
            ↪  sortedset
                ss.remove(nums[i-k])
                n -= 1
        return False
```

 系列题，关于 longest substring distinct characters 很多类似的题目，总结一下！主要是 hashtable, sliding window 的结合，
↪ 复杂的 case 需要 dp. upstart 考了类似的题目！

 substring, subarray, subsequence 三种常见的问题，总结一下！

 Substring & string 类型的

159. Longest Substring with At Most Two Distinct Characters
↪ https://leetcode.com/problems/longest-substring-with-at-most-two-distinct-characters/

Given a string s, return the length of the longest substring that contains at most two distinct characters.
Example 1:
```
Input: s = "eceba"
```
```

```
Output: 3
Explanation: The substring is "ece" which its length is 3.
```
Example 2:
```
Input: s = "ccaabbb"
Output: 5
Explanation: The substring is "aabbb" which its length is 5.
```

```python
# 经典 sliding window 模板题,
class Solution:
    def lengthOfLongestSubstringTwoDistinct(self, s: str) -> int:
        # 这个模板不错!
        slow = 0
        n = len(s)
        # hashset = set() # 尽量用 set 来弄, 或者 hashmap
        hashmap = {}
        res = 0
        for fast in range(n):
            # 先去操作目标, 放进 hashmap
            hashmap[s[fast]] = hashmap.get(s[fast],0) + 1
            # 先判断是否满足条件, 如果满足, 操作
            if len(hashmap) <= 2:
                res = max(res, fast - slow + 1)
            # 不满足的话, 想办法更新 slow 指针
            while len(hashmap) > 2:
                head = s[slow]
                hashmap[head] -= 1
                if hashmap[head] == 0:
                    del hashmap[head]
                slow += 1

        return res
```

340. Longest Substring with At Most K Distinct Characters
↪ https://leetcode.com/problems/longest-substring-with-at-most-k-distinct-characters/
Given a string s and an integer k, return the length of the longest substring of s that contains at most k
↪ distinct characters.
```
Example 1:

Input: s = "eceba", k = 2
Output: 3
Explanation: The substring is "ece" with length 3.
Example 2:

Input: s = "aa", k = 1
Output: 2
Explanation: The substring is "aa" with length 2.
```

```
> 这个题如果难一点就是让返回所有的最长的 substring 这个要自己写一下

```python
class Solution:
    def lengthOfLongestSubstringKDistinct(self, s: str, k: int) -> int:
        # 模板确实厉害! 清晰
        slow, hashm = 0, {}
        res = 0
        for fast in range(len(s)):
            tail = s[fast]
            hashm[tail] = hashm.get(tail, 0) + 1
            if len(hashm) <= k:
                res = max(res, fast - slow + 1)
            while len(hashm) > k: # 这个位置就是想清楚, 不满足 if 条件, 用 if 还是 while
                head = s[slow]
                hashm[head] -= 1
                if hashm[head] == 0:
                    del hashm[head]
                slow += 1
        return res
```

3. Longest Substring Without Repeating Characters
↪ https://leetcode.com/problems/longest-substring-without-repeating-characters/
Given a string s, find the length of the longest substring without repeating characters.
```
Example 1:

Input: s = "abcabcbb"
Output: 3
Explanation: The answer is "abc", with the length of 3.
Example 2:

Input: s = "bbbbb"
Output: 1
Explanation: The answer is "b", with the length of 1.
Example 3:

Input: s = "pwwkew"
Output: 3
Explanation: The answer is "wke", with the length of 3.
Notice that the answer must be a substring, "pwke" is a subsequence and not a substring.
```
> 这个和 340 非常一致, 可以放到一起来做, 一个 follow up 是返回所有的最长的 substring! upstart 考了!

```python
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        # 这个模板还是不错的!
        slow = 0
        hashmap = {}
```
```

```
        res = 0
        n = len(s)
        for fast in range(n):
            tail = s[fast]
            hashmap[tail] = hashmap.get(tail, 0) + 1
            if len(hashmap) == fast - slow + 1:
                res = max(res, fast - slow + 1)
            while fast - slow + 1 > len(hashmap):
                head = s[slow]
                hashmap[head] -= 1
                if hashmap[head] == 0:
                    del hashmap[head]
                slow += 1
        return res
```

> Follow-up: return all 最长的 substring

Example

```
input = "fsfetwenwac"
output = ['sfetw', 'enwac']
```

```python
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        # 这个模板还是不错的!
        slow = 0
        hashmap = {}
        max_len = 0
        n = len(s)
        res_list = []
        for fast in range(n):
            tail = s[fast]
            hashmap[tail] = hashmap.get(tail, 0) + 1
            if len(hashmap) == fast - slow + 1:
                # 这段是核心记录所有 list 的办法! 其实不难
                print(fast - slow + 1, max_len)
                if fast - slow + 1 > max_len:
                    res_list = [] # 这个就是通过 [] 来不断 update
                    res_list.append((slow, fast))
                    print(res_list)
                    max_len = max(max_len, fast - slow + 1)
                elif fast - slow + 1 == max_len:
                    res_list.append((slow, fast))

            while fast - slow + 1 > len(hashmap):
                head = s[slow]
                hashmap[head] -= 1
                if hashmap[head] == 0:
                    del hashmap[head]
                slow += 1
        # 用 tuple 记录 slow and fast 位置, 然后最后一起输出
        output = []
```

```
        print(res_list)
        for i, j in res_list:
            output.append(s[i:j+1])

        return output
```

> 395 算是 substring，但不算典型的 sliding window

395. Longest Substring with At Least K Repeating Characters
https://leetcode.com/problems/longest-substring-with-at-least-k-repeating-characters/

Given a string s and an integer k, return the length of the longest substring of s such that the frequency of each character in this substring is greater than or equal to k.

```
Example 1:

Input: s = "aaabb", k = 3
Output: 3
Explanation: The longest substring is "aaa", as 'a' is repeated 3 times.
Example 2:

Input: s = "ababbc", k = 2
Output: 5
Explanation: The longest substring is "ababb", as 'a' is repeated 2 times and 'b' is repeated 3 times.
```

> 这个用 brute force 可以，但是 sliding window 很难写，关键是如何控制 window 里面的最小值，递归的方法不好想，不具有通用性！虽然看上去和 340 挺像，但实际上完全不一样!!!!

```python
class Solution:
    def longestSubstring(self, s: str, k: int) -> int:
        # 这个题用正常的 sliding window 很难做!
        # brute force 要想出来 double for loop - time o(n^2), space o(1)
        n = len(s)
        res = 0
        for i in range(n):
            for j in range(i + 1, n + 1): # 注意这个地方要 n+1
                hashmap = Counter(s[i:j]) # 这个也要注意
                if min(hashmap.values()) >= k: # 这个是可以操作的
                    res = max(res, j - i)
        return res

# # 递归的解法有点秀!
# class Solution(object):
#     def longestSubstring(self, s, k):
#         if len(s) < k:
#             return 0
#         for c in set(s):
#             if s.count(c) < k:
#                 return max(self.longestSubstring(t, k) for t in s.split(c))
#         return len(s)
```

424. Longest Repeating Character Replacement https://leetcode.com/problems/longest-repeating-character-replacement/

You are given a string s and an integer k. You can choose any character of the string and change it to any other uppercase English character. You can perform this operation at most k times. Return the length of the longest substring containing the same letter you can get after performing the above operations.

```
Example 1:

Input: s = "ABAB", k = 2
Output: 4
Explanation: Replace the two 'A's with two 'B's or vice versa.
Example 2:

Input: s = "AABABBA", k = 1
Output: 4
Explanation: Replace the one 'A' in the middle with 'B' and form "AABBBBA".
The substring "BBBB" has the longest repeating letters, which is 4.
```

> 不算 substring，但是 sliding windows 相关，有一类题就是可以替换操作!

```python
class Solution:
    def characterReplacement(self, s: str, k: int) -> int:
        # 应用模板不错的一个题
        slow, res, max_freq, hashm = 0, 0, 0, {}
        for fast in range(len(s)):
            tail = s[fast]
            hashm[tail] = hashm.get(tail, 0) + 1
            max_freq = max(max_freq, hashm[tail]) # 这是关键，统计 frequency，和 01 问题的区别，当时给定 1 了
            if fast - slow + 1 <= max_freq + k: # 这步必须是 <= 之前也有类似的问题
                res = max(res, fast - slow + 1)
            while fast - slow + 1 > max_freq + k:
                head = s[slow]
                hashm[head] -= 1
                if hashm[head] == 0:
                    del hashm[head]
                slow += 1
        return res
```

438. Find All Anagrams in a String https://leetcode.com/problems/find-all-anagrams-in-a-string/

Given two strings s and p, return an array of all the start indices of p's anagrams in s. You may return the answer in any order.

An Anagram is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

```
Example 1:

Input: s = "cbaebabacd", p = "abc"
Output: [0,6]
Explanation:
The substring with start index = 0 is "cba", which is an anagram of "abc".
```

```
The substring with start index = 6 is "bac", which is an anagram of "abc".
Example 2:

Input: s = "abab", p = "ab"
Output: [0,1,2]
Explanation:
The substring with start index = 0 is "ab", which is an anagram of "ab".
The substring with start index = 1 is "ba", which is an anagram of "ab".
The substring with start index = 2 is "ab", which is an anagram of "ab".
```
> sliding window 模板题
```python
class Solution:
    def findAnagrams(self, s: str, p: str) -> List[int]:
        # 模板确实不错!
        res = []
        slow = 0
        hash_s = {}
        hash_p = {}
        for char in p:
            hash_p[char] = hash_p.get(char, 0) + 1
        for fast in range(len(s)):
            hash_s[s[fast]] = hash_s.get(s[fast], 0) + 1
            if hash_s == hash_p:
                res.append(slow)
            if fast >= len(p) - 1:
                head = s[slow]
                hash_s[head] -= 1
                if hash_s[head] == 0:
                    del hash_s[head]
                slow += 1
        return res
```

567. Permutation in String https://leetcode.com/problems/permutation-in-string/
Given two strings s1 and s2, return true if s2 contains a permutation of s1, or false otherwise. In other words, return true if one of s1's permutations is the substring of s2.
```
Example 1:

Input: s1 = "ab", s2 = "eidbaooo"
Output: true
Explanation: s2 contains one permutation of s1 ("ba").
Example 2:

Input: s1 = "ab", s2 = "eidboaoo"
Output: false
```
> 和 438 很像主席细节
```python
class Solution:
    def checkInclusion(self, s1: str, s2: str) -> bool:
```

```
        # sliding window 的题，和 438 非常像，本身也算 permutation
        hashs1 = {}
        hashs2 = {}
        for char in s1:
            hashs1[char] = hashs1.get(char, 0) + 1

        slow = 0
        for fast in range(len(s2)):
            hashs2[s2[fast]] = hashs2.get(s2[fast], 0) + 1
            if hashs2 == hashs1:
                return True
            if fast >= len(s1) - 1:
                head = s2[slow] # 注意细节，不是一味的背模板!
                hashs2[head] -= 1
                if hashs2[head] == 0:
                    del hashs2[head]
                slow +=1

        return False
```


□□□□ 系列题，max consecutive ones 1,2,3

485. Max Consecutive Ones https://leetcode.com/problems/max-consecutive-ones/

Given a binary array nums, return the maximum number of consecutive 1's in the array.
```
Example 1:

Input: nums = [1,1,0,1,1,1]
Output: 3
Explanation: The first two digits or the last three digits are consecutive 1s. The maximum number of consecutive
↪  1s is 3.
Example 2:

Input: nums = [1,0,1,1,0,1]
Output: 2
```
```python
class Solution:
    def findMaxConsecutiveOnes(self, nums: List[int]) -> int:
        temp = 0
        res = 0
        for i in nums:
            if i == 1:
                temp += 1
            else:
                temp = 0
            res = max(res, temp)

        return res
```

```

### 487. Max Consecutive Ones II  https://leetcode.com/problems/max-consecutive-ones-ii/
Given a binary array nums, return the maximum number of consecutive 1's in the array if you can flip at most one 0.
```
Example 1:

Input: nums = [1,0,1,1,0]
Output: 4
Explanation: Flip the first zero will get the maximum number of consecutive 1s. After flipping, the maximum number of consecutive 1s is 4.
Example 2:

Input: nums = [1,0,1,1,0,1]
Output: 4
```
> 比 485 复杂但基本也是模板

```python
class Solution:
    def findMaxConsecutiveOnes(self, nums: List[int]) -> int:
        # get(key) 方法在 key (键) 不在字典中时，可以返回默认值 None 或者设置的默认值
        # dict[key] 在 key (键) 不在字典中时，会触发 KeyError 异常。
        slow = 0
        num_zero = 0
        hashm = {}
        res = 0
        for fast in range(len(nums)):
            tail = nums[fast]
            hashm[tail] = hashm.get(tail, 0) + 1
            if hashm.get(0, 0) <= 1:  # 如果直接 call dict[key] 就会报错，因为没有 0，可能
                res = max(res, fast - slow + 1)

            while hashm.get(0, 0) > 1: # 如果直接 call dict[key] 就会报错，因为没有 0，可能
                head = nums[slow]
                hashm[head] -= 1
                if hashm[head] == 0:
                    del hashm[head]

                slow += 1
        return res
```

1004. Max Consecutive Ones III https://leetcode.com/problems/max-consecutive-ones-iii/

Given a binary array nums and an integer k, return the maximum number of consecutive 1's in the array if you can flip at most k 0's.
```
Example 1:

```
```

```
Input: nums = [1,1,1,0,0,0,1,1,1,1,0], k = 2
Output: 6
Explanation: [1,1,1,0,0,1,1,1,1,1,1]
Bolded numbers were flipped from 0 to 1. The longest subarray is underlined.
Example 2:

Input: nums = [0,0,1,1,0,0,1,1,1,0,1,1,0,0,0,1,1,1,1], k = 3
Output: 10
Explanation: [0,0,1,1,1,1,1,1,1,1,1,1,0,0,0,1,1,1,1]
Bolded numbers were flipped from 0 to 1. The longest subarray is underlined.
```
> 和 487 完全一样，从 1 变成 k，这个模板不错!

```python
class Solution:
    def longestOnes(self, nums: List[int], k: int) -> int:
        slow = 0
        num_zero = 0
        hashm = {}
        res = 0
        for fast in range(len(nums)):
            tail = nums[fast]
            hashm[tail] = hashm.get(tail, 0) + 1
            if hashm.get(0, 0) <= k:  # 如果直接 call dict[key] 就会报错，因为没有 0，可能
                res = max(res, fast - slow + 1)

            while hashm.get(0, 0) > k: # 如果直接 call dict[key] 就会报错，因为没有 0，可能
                head = nums[slow]
                hashm[head] -= 1
                if hashm[head] == 0:
                    del hashm[head]

                slow += 1
        return res
```

1446. Consecutive Characters https://leetcode.com/problems/consecutive-characters/
The power of the string is the maximum length of a non-empty substring that contains only one unique character.Given a string s, return the power of s.
```
Example 1:

Input: s = "leetcode"
Output: 2
Explanation: The substring "ee" is of length 2 with the character 'e' only.
Example 2:

Input: s = "abbcccddddeeeeedcba"
Output: 5
Explanation: The substring "eeeee" is of length 5 with the character 'e' only.
```
> one pass 这种相邻的题目是一类题，substring 的这个是最简单的!

```python
class Solution:
    def maxPower(self, s: str) -> int:
        res = 1 # 这个初始化是 1 不是 0
        maxtemp = 1
        for i in range(1, len(s)):
            if s[i] == s[i-1]:
                maxtemp += 1
                res = max(res, maxtemp)
            else:
                maxtemp = 1
        return res
        # time o(n), space o(1)
```

🔥🔥🔥 Subarray 题型总结!
> 有一类就是和 K 结合, product less than K, summary less than K

643. Maximum Average Subarray I https://leetcode.com/problems/maximum-average-subarray-i/
You are given an integer array nums consisting of n elements, and an integer k. Find a contiguous subarray whose length is equal to k that has the maximum average value and return this value. Any answer with a calculation error less than 10-5 will be accepted.
```
Example 1:

Input: nums = [1,12,-5,-6,50,3], k = 4
Output: 12.75000
Explanation: Maximum average is (12 - 5 - 6 + 50) / 4 = 51 / 4 = 12.75
Example 2:

Input: nums = [5], k = 1
Output: 5.00000
```
> subarray 经典入门题
```python
class Solution:
    def findMaxAverage(self, nums: List[int], k: int) -> float:
        slow = 0
        n = len(nums)
        sum_ = 0
        res = -inf
        for fast in range(n):
            sum_ += nums[fast]
            # 不满足窗口条件, 对 slow 操作, 有时候也用 while ? 因为这个是 fix 窗口, 所以用 if
            if fast >= k - 1:
                sum_ -= nums[slow]
                slow += 1
            # 这步可以理解! 满足条件然后操作
            if fast - slow + 1 == k:
                res = max(res, sum_ / k)
        return res
```

644. Maximum Average Subarray II https://leetcode.com/problems/maximum-average-subarray-ii/ □□□

You are given an integer array nums consisting of n elements, and an integer k. Find a contiguous subarray whose length is greater than or equal to k that has the maximum average value and return this value. Any answer with a calculation error less than 10-5 will be accepted.

```
Example 1:

Input: nums = [1,12,-5,-6,50,3], k = 4
Output: 12.75000
Explanation:
- When the length is 4, averages are [0.5, 12.75, 10.5] and the maximum average is 12.75
- When the length is 5, averages are [10.4, 10.8] and the maximum average is 10.8
- When the length is 6, averages are [9.16667] and the maximum average is 9.16667
The maximum average is when we choose a subarray of length 4 (i.e., the sub array [12, -5, -6, 50]) which has the max average 12.75, so we return 12.75
Note that we do not consider the subarrays of length < 4.
Example 2:

Input: nums = [5], k = 1
Output: 5.00000
```

> 虽然是 643 相似，但这个题是二分法

```python
class Solution:
    def findMaxAverage(self, nums: List[int], k: int) -> float:
        if not nums:
            return 0
        start, end = min(nums), max(nums)
        while end - start > 1e-5:
            mid = (start + end) / 2
            if self.check_subarray(nums, k, mid):
                start = mid
            else:
                end = mid
        return start
    def check_subarray(self, nums, k, average):
        prefix_sum = [0]
        for num in nums:
            prefix_sum.append(prefix_sum[-1] + num - average)

        min_prefix_sum = 0
        for i in range(k, len(nums) + 1):
            if prefix_sum[i] - min_prefix_sum >= 0:
                return True
            min_prefix_sum = min(min_prefix_sum, prefix_sum[i - k + 1])
        return False
```

713. Subarray Product Less Than K https://leetcode.com/problems/subarray-product-less-than-k/

Given an array of integers nums and an integer k, return the number of contiguous subarrays where the product of all the elements in the subarray is strictly less than k.
```
Example 1:

Input: nums = [10,5,2,6], k = 100
Output: 8
Explanation: The 8 subarrays that have product less than 100 are:
[10], [5], [2], [6], [10, 5], [5, 2], [2, 6], [5, 2, 6]
Note that [10, 5, 2] is not included as the product of 100 is not strictly less than k.
Example 2:

Input: nums = [1,2,3], k = 0
Output: 0
```
> 之前的模板要改，不能直接用，因为先判断的话，其实 window 并不合法，所以要在最后存结果
```python
class Solution:
    def numSubarrayProductLessThanK(self, nums: List[int], k: int) -> int:
        # 这个题不错，能不能输出所有的 pair
        if k <= 1: return 0
        slow = 0
        prod = 1
        res = 0
        for fast in range(len(nums)):
            prod *= nums[fast]
            # if prod < k: 都是错误的，不能在这存结果，跟之前的模板不同，这个 window 不合法!
            #     res += fast - slow + 1  # 放在这就错误了，没有更新 slow
            while prod >= k:
                prod /= nums[slow]
                slow += 1
            res += fast - slow + 1
        return res
        # time o(n), space o(1)

        # brute force # time o(n^2) and space o(1) 会超时
        # res = 0
        # for i in range(len(nums)):
        #     prod = 1
        #     for j in range(i, len(nums)):
        #         prod *= nums[j]
        #         if prod < k:
        #             res += 1
        #         else:
        #             continue
        # return res
        # time o(n^2), space o(1)
```
Follow-up: 如何输出所有符合的 subarray # 如果要输出所有的 subarrays 相当于在 nums[slow:fast] 这个 window 里的 subset 所有合集? 不好做!

🔥🔥🔥 Subsequence 的类型题！很多要用 DP？

674. Longest Continuous Increasing Subsequence
https://leetcode.com/problems/longest-continuous-increasing-subsequence/

Given an unsorted array of integers nums, return the length of the longest continuous increasing subsequence (i.e. subarray). The subsequence must be strictly increasing. A continuous increasing subsequence is defined by two indices l and r (l < r) such that it is [nums[l], nums[l + 1], ..., nums[r - 1], nums[r]] and for each l <= i < r, nums[i] < nums[i + 1].

```
Example 1:

Input: nums = [1,3,5,4,7]
Output: 3
Explanation: The longest continuous increasing subsequence is [1,3,5] with length 3.
Even though [1,3,5,7] is an increasing subsequence, it is not continuous as elements 5 and 7 are separated by element
4.
Example 2:

Input: nums = [2,2,2,2,2]
Output: 1
Explanation: The longest continuous increasing subsequence is [2] with length 1. Note that it must be strictly
increasing.
```

> 这个题挺不错，很多种解法！

```python
class Solution:
    def findLengthOfLCIS(self, nums: List[int]) -> int:

        # 这个题虽然是一个 easy，但是很多种解法，很多不错的方法！
        # way 1 - 上来想到的是 simulate 也可以叫 greey？
        # 这里比较的是 i + 1 和 i，然后从 len(nums) - 1 开始的
#         if len(nums) == 0: return 0
#         res = 1
#         count = 1
#         for i in range(len(nums) - 1):
#             if nums[i + 1] > nums[i]:
#                 count += 1
#             else:
#                 count = 1
#             res = max(res, count)
#         return res

        # # 这个写法也可以，从 1 开始的，之前写的有问题
        # res = 1
        # count = 1
        # for i in range(1, len(nums)):
        #     if nums[i] > nums[i - 1]:
        #         count += 1
        #     else:
```

```
        #         count = 1
        #     res = max(res, count)
        # return res


        # way 2 - sliding window, double pointer 这里面 sliding window 有 2 种可以做!
        # 方法 1 是锚钉, 像 solution 说的那种
        # res, anchor = 0, 0
        # for i in range(len(nums)):
        #     if i and nums[i - 1] >= nums[i]:
        #         anchor = i
        #     res = max(res, i - anchor + 1) # 在出现 nums[i - 1] >= nums[i] 之前, anchor 总是 0, 没有更新, 所以这个就是记录的最大值
        # return res


        # # 方法 2 是滑动 while 然后求最大长度 这个理解的不错! 复习一下 sliding windows!
        # if not nums: return 0
        # slow, fast = 0, 1
        # res = 1
        # n = len(nums)
        # for fast in range(n):
        #     while fast < n and nums[fast] > nums[fast - 1]:
        #         fast += 1
        #     res = max(res, fast - slow)
        #     slow = fast
        # return  res
        # # time 一样是 o(n) and space is o(1)

        # way 3 - DP 方法!
        # 确定 dp 的含义 dp[i] 以下标 i 为结尾的数组的连续递增子序列长度
        if len(nums) == 0: return 0
        dp = [1] * len(nums)
        res = 1
        for i in range(len(nums) - 1):
            if nums[i + 1] > nums[i]:
                dp[i + 1] = dp[i] + 1
            res = max(res, dp[i + 1])
        return res
```

🔥🔥🔥 hard 题 sliding window 🔥🔥🔥

76. Minimum Window Substring https://leetcode.com/problems/minimum-window-substring/

Given two strings s and t of lengths m and n respectively, return the minimum window substring of s such that every character in t (including duplicates) is included in the window. If there is no such substring, return the empty string "". The testcases will be generated such that the answer is unique. A substring is a contiguous sequence of characters within the string.

```
Example 1:

Input: s = "ADOBECODEBANC", t = "ABC"
Output: "BANC"
Explanation: The minimum window substring "BANC" includes 'A', 'B', and 'C' from string t.
Example 2:

Input: s = "a", t = "a"
Output: "a"
Explanation: The entire string s is the minimum window.
Example 3:

Input: s = "a", t = "aa"
Output: ""
Explanation: Both 'a's from t must be included in the window.
Since the largest window of s only has one 'a', return empty string.
```

> 很复杂，不会

```python
class Solution:
    def minWindow(self, s: str, t: str) -> str:

        # write your code here
        target, source = t, s

        if len(target) == 0 or len(source) == 0:
            return ''

        m, n = len(target), len(source)
        target_c, sub_c = {}, {}

        for i in range(m):
            target_c[target[i]] = target_c.get(target[i], 0) + 1

        fast = 0
        matched_chars = 0
        start, substring_len = 0, float('inf')

        for slow in range(n):

            while fast < n and matched_chars < len(target_c):
                sub_c[source[fast]] = sub_c.get(source[fast], 0) + 1
                if sub_c[source[fast]] == target_c.get(source[fast], 0):
```

```
                    matched_chars += 1
                fast += 1

            if matched_chars == len(target_c):
                if substring_len > fast - slow:
                    substring_len = fast - slow
                    start = slow

            sub_c[source[slow]] -= 1
            if sub_c[source[slow]] == target_c.get(source[slow], 0) - 1:
                matched_chars -= 1

        if substring_len == float('inf') :
            return ''

        return source[start : start + substring_len]
```

239. Sliding Window Maximum https://leetcode.com/problems/sliding-window-maximum/

You are given an array of integers nums, there is a sliding window of size k which is moving from the very left of the array to the very right. You can only see the k numbers in the window. Each time the sliding window moves right by one position. Return the max sliding window.

```
Example 1:
Input: nums = [1,3,-1,-3,5,3,6,7], k = 3
Output: [3,3,5,5,6,7]
Explanation:
Window position                Max
---------------               -----
[1  3  -1] -3  5  3  6  7       3
 1 [3  -1  -3] 5  3  6  7       3
 1  3 [-1  -3  5] 3  6  7       5
 1  3  -1 [-3  5  3] 6  7       5
 1  3  -1  -3 [5  3  6] 7       6
 1  3  -1  -3  5 [3  6  7]      7
Example 2:
Input: nums = [1], k = 1
Output: [1]
```

> hard 不会，没有做出来，跳过

```python
class Solution:
    def maxSlidingWindow(self, nums: List[int], k: int) -> List[int]:

        n = len(nums)
        q = collections.deque()
        for i in range(k):
            while q and nums[i] >= nums[q[-1]]:
                q.pop()
            q.append(i)
```

```
        ans = [nums[q[0]]]
        for i in range(k, n):
            while q and nums[i] >= nums[q[-1]]:
                q.pop()
            q.append(i)
            while q[0] <= i - k:
                q.popleft()
            ans.append(nums[q[0]])

        return ans
```
30. Substring with Concatenation of All Words
https://leetcode.com/problems/substring-with-concatenation-of-all-words/ [hard]

You are given a string s and an array of strings words of the same length. Return all starting indices of substring(s) in s that is a concatenation of each word in words exactly once, in any order, and without any intervening characters. You can return the answer in any order.

Example 1:
```
Input: s = "barfoothefoobarman", words = ["foo","bar"]
Output: [0,9]
Explanation: Substrings starting at index 0 and 9 are "barfoo" and "foobar" respectively.
The output order does not matter, returning [9,0] is fine too.
```
Example 2:
```
Input: s = "wordgoodgoodgoodbestword", words = ["word","good","best","word"]
Output: []
```
Example 3:
```
Input: s = "barfoofoobarthefoobarman", words = ["bar","foo","the"]
Output: [6,9,12]
```

```python
# 完全不会，太难!
class Solution:
    def findSubstring(self, s: str, words: List[str]) -> List[int]:
        from collections import Counter
        if not s or not words:return []
        one_word = len(words[0])
        word_num = len(words)
        n = len(s)
        words = Counter(words)
        res = []
        for i in range(0, one_word):
            cur_cnt = 0
            left = i
            right = i
            cur_Counter = Counter()
            while right + one_word <= n:
```

```
                w = s[right:right + one_word]
                right += one_word
                cur_Counter[w] += 1
                cur_cnt += 1
                while cur_Counter[w] > words[w]:
                    left_w = s[left:left+one_word]
                    left += one_word
                    cur_Counter[left_w] -= 1
                    cur_cnt -= 1
                if cur_cnt == word_num :
                    res.append(left)
        return res
```

---

3 Binary search

4 Recursion backtracking

5 Graph DFS

6 Graph BFS

7 Binary Tree

8 Stack

9 Heap (Priority Queue)

```
# heap

## 定义
1. 分为最小堆 minheap, 和最大堆, maxheap, 也就是最小元素或者最大元素在堆顶
2. 堆是一个完全二叉树, 但堆的底层实现一般是数组, 而不是二叉树
3. 孩子节点都比父亲节点大, 但是左右孩子的大小不影响
4. 堆不是 binary search tree
5. 堆的操作是从上到下, 从左到右

## 基本操作 - 高度是 logn
1. 构建堆 heapify - o(n)
2. 遍历堆 o(nlogn)
3. add - o(n)
4. remove - 理论上是 o(logn) 但实际上 python 的库函数是 for loop 遍历的, 所以是 o(n)
5. pop - o(logn),  push 也是 o(logn)
6. min or max - o(1)
7. 由于是数组操作, 选定 k, 父亲是 k/2, 左孩子 kx2, 右孩子 kx2+1
8. 可以结合 hashmap 去查询或者 remove 指定值
```

使用条件
1. 找最大值或者最小值 (60%)
2. 找第 k 大 (pop k 次 复杂度 o(nlogk)) (50%)
3. 要求 logn 时间对数据进行操作 (40%)

堆不能解决的问题
1. 查询比某个数大的最小值或者最近接的值 (平衡二叉树 balanced bst 才可以解决)
2. 找某段区间的最大值最小值 (线段树 segmenttree 可以解决)
3. o(n) 找第 k 大的数 (需要使用快排中的 partition 操作)

time complexity
space complexity

template
```python
from heapq import heappush, heappop

class Heap:
    def __init__(self):
        self.minheap()
        self.deleted_set = set()

    def push(self, index, val):
        heappush(self.minheal, (val, index))

    def _lazy_deletion(self):
        while self.minheap and self.minheap[0][1] in self.deleted_set:
            heappop(self.minheap)

    def top(self):
        self._lazy_deletion()
        return self.minheap[0]

    def pop(self): # 移除顶端元素
        self._lazy_deletion()
        heappop(self.minheap)

    def delete(self, index):
        self.deleted_set.add(index)

    def is_empty(self):
        return not bool(self.minheap)

```

几种类型的双指针及相关题目

Example 264. Ugly Number II
https://leetcode.com/problems/ugly-number-ii/

An ugly number is a positive integer whose prime factors are limited to 2, 3, and 5.

Given an integer n, return the nth ugly number.

Example 1:
```
Input: n = 10
Output: 12
Explanation: [1, 2, 3, 4, 5, 6, 8, 9, 10, 12] is the sequence of the first 10 ugly numbers.
```
Example 2:
```
Input: n = 1
Output: 1
Explanation: 1 has no prime factors, therefore all of its prime factors are limited to 2, 3, and 5.
```
>solution
```python
        # 用 heap 来做，不断的找最小值，然后 push 回去，时间复杂度 o(nlogn)

        import heapq

        heap = [1]
        visited = set([1])

        min_val = None
        for i in range(n): # n 次操作，也是 nth 的最小值
            min_val = heapq.heappop(heap)
            for factor in [2, 3, 5]:
                if min_val * factor not in visited:
                    visited.add(min_val * factor)
                    heapq.heappush(heap, min_val * factor)

        return min_val
```

973. K Closest Points to Origin
https://leetcode.com/problems/k-closest-points-to-origin/

Given an array of points where points[i] = [xi, yi] represents a point on the X-Y plane and an integer k, return the k closest points to the origin (0, 0).

The distance between two points on the X-Y plane is the Euclidean distance (i.e., √(x1 - x2)2 + (y1 - y2)2).

You may return the answer in any order. The answer is guaranteed to be unique (except for the order that it is in).

Example 1:
```
```

```
Input: points = [[1,3],[-2,2]], k = 1
Output: [[-2,2]]
Explanation:
The distance between (1, 3) and the origin is sqrt(10).
The distance between (-2, 2) and the origin is sqrt(8).
Since sqrt(8) < sqrt(10), (-2, 2) is closer to the origin.
We only want the closest k = 1 points from the origin, so the answer is just [[-2,2]].
```
Example 2:
```
Input: points = [[3,3],[5,-1],[-2,4]], k = 2
Output: [[3,3],[-2,4]]
Explanation: The answer [[-2,4],[3,3]] would also be accepted.
```

>solution

```python
#         # heap  最小堆 - 把所有点都放入最小堆，然后用最小堆取出 k 个，时间 o(nlogn) 空间 o(n) + o(k)
#         # 遍历堆 o(nlogn) 所以时间是这个 level，空间是因为开了 heap + res 的部分，单独储存 res


#         import heapq
#         heap = []

#         for point in points:
#             cur_dis = point[0] ** 2 + point[1] ** 2
#             heapq.heappush(heap, (cur_dis, point)) # 注意加进去的时候还是看 cur_dis 自动排序好了

#         res = []
#         i = 0
#         while i < k:
#             _, point = heapq.heappop(heap)
#             res.append(point)
#             i += 1

        # return res

        # 最大堆 更优一些，因为不需要重新开整个 heap - 时间 o(nlogk)，空间 o(k)
        heap = []
        for point in points:
            cur_dis = point[0] ** 2 + point[1] ** 2
            heapq.heappush(heap, (-cur_dis, point)) # 注意加进去的时候还是看 cur_dis 自动排序好了
            if len(heap) > k:
                heapq.heappop(heap)

        res = []
        i = 0
        while i < k:
            _, point = heapq.heappop(heap)
            res.append(point)
            i += 1
```

```
        return res
```

Example: Lintcode 545 · Top k Largest Numbers II
https://www.lintcode.com/problem/545/

Description
Implement a data structure, provide two interfaces:

add(number). Add a new number in the data structure.
topk(). Return the top k largest numbers in this data structure. k is given when we create the data structure.

>solution
```python
import heapq
class Solution:
    """
    @param: k: An integer
    """
    def __init__(self, k):
        # do intialization if necessary
        self.k = k
        self.heap = []

    """
    @param: num: Number to be added
    @return: nothing
    """
    def add(self, num):
        # write your code here
        heapq.heappush(self.heap, num)
        if len(self.heap) > self.k:
            heapq.heappop(self.heap)


    """
    @return: Top k element
    """
    def topk(self):
        # write your code here
        return sorted(self.heap, reverse=True)
```

253. Meeting Rooms II
https://leetcode.com/problems/meeting-rooms-ii/

Given an array of meeting time intervals intervals where intervals[i] = [starti, endi], return the minimum number of conference rooms required.

Example 1:
```
```

```
Input: intervals = [[0,30],[5,10],[15,20]]
Output: 2
```
Example 2:
```
Input: intervals = [[7,10],[2,4]]
Output: 1
```

>solution
https://leetcode.com/problems/meeting-rooms-ii/solution/

```python
class Solution:
    def minMeetingRooms(self, intervals: List[List[int]]) -> int:

        # If there is no meeting to schedule then no room needs to be allocated.
        if not intervals:
            return 0

        # The heap initialization
        free_rooms = []

        # Sort the meetings in increasing order of their start time.
        intervals.sort(key= lambda x: x[0])

        # Add the first meeting. We have to give a new room to the first meeting.
        heapq.heappush(free_rooms, intervals[0][1])

        # For all the remaining meeting rooms
        for i in intervals[1:]:

            # If the room due to free up the earliest is free, assign that room to this meeting.
            if free_rooms[0] <= i[0]:
                heapq.heappop(free_rooms)

            # If a new room is to be assigned, then also we add to the heap,
            # If an old room is allocated, then also we have to add to the heap with updated end time.
            heapq.heappush(free_rooms, i[1])

        # The size of the heap tells us the minimum rooms required for all the meetings.
        return len(free_rooms)

# Complexity Analysis

# Time Complexity: O(NlogN)

# There are two major portions that take up time here. One is sorting of the array that takes O(NlogN)
↪  considering that the array consists of NN elements.Then we have the min-heap. In the worst case, all NN
↪  meetings will collide with each other. In any case we have NN add operations on the heap. In the worst case
↪  we will have NN extract-min operations as well. Overall complexity being (NlogN)(NlogN) since extract-min
↪  operation on a heap takes O(logN).
```

```
# Space Complexity: O(N) because we construct the min-heap and that can contain NN elements in the worst case as described above in the time complexity section. Hence, the space complexity is O(N).
```

Example: 373. Find K Pairs with Smallest Sums
https://leetcode.com/problems/find-k-pairs-with-smallest-sums/

You are given two integer arrays nums1 and nums2 sorted in ascending order and an integer k.

Define a pair (u, v) which consists of one element from the first array and one element from the second array.

Return the k pairs (u1, v1), (u2, v2), ..., (uk, vk) with the smallest sums.

Example 1:
```
Input: nums1 = [1,7,11], nums2 = [2,4,6], k = 3
Output: [[1,2],[1,4],[1,6]]
Explanation: The first 3 pairs are returned from the sequence: [1,2],[1,4],[1,6],[7,2],[7,4],[11,2],[7,6],[11,4],[11,6]
```
Example 2:
```
Input: nums1 = [1,1,2], nums2 = [1,2,3], k = 2
Output: [[1,1],[1,1]]
Explanation: The first 2 pairs are returned from the sequence: [1,1],[1,1],[1,2],[2,1],[1,2],[2,2],[1,3],[1,3],[2,3]
```
Example 3:
```
Input: nums1 = [1,2], nums2 = [3], k = 3
Output: [[1,3],[2,3]]
Explanation: All possible pairs are returned from the sequence: [1,3],[2,3]
```

>solution
```python
# 用 heap 来解 - 和 973 很像，但是是二维遍历，然后也是用最大堆
        import heapq
        heap = []
        for i in range(min(k, len(nums1))):
            for j in range(min(k, len(nums2))):
                if len(heap) < k:
                    heapq.heappush(heap, ((-nums1[i] - nums2[j]), i, j))
                else:
                    if nums1[i] + nums2[j] < -heap[0][0]:
                        heappop(heap)
                        heappush(heap, (-(nums1[i] + nums2[j]), i, j))
        res = []
        for _,i,j in heap:
            res.append( [nums1[i], nums2[j]])
```

```
        return res

        # time - klogk? 不太确定
        # space - o(k)
```

Example: 215. Kth Largest Element in an Array
https://leetcode.com/problems/kth-largest-element-in-an-array/
Given an integer array nums and an integer k, return the kth largest element in the array.

Note that it is the kth largest element in the sorted order, not the kth distinct element.

Example 1:
```
Input: nums = [3,2,1,5,6,4], k = 2
Output: 5
```
Example 2:
```
Input: nums = [3,2,3,1,2,4,5,5,6], k = 4
Output: 4
```

>solution
```python
        # 暴力干 - 但没有什么意思， python 的 sort 时间复杂度是多少？不清楚
        # nums.sort()
        # return nums[-k]

# That would be an algorithm of O(NlogN) time complexity and
# O(1) space complexity.

        # heap
        import heapq
        return heapq.nlargest(k, nums)[-1]

# Time complexity : O(Nlogk).
# Space complexity : O(k) to store the heap elements.
```


Example : 692. Top K Frequent Words
https://leetcode.com/problems/top-k-frequent-words/

Given an array of strings words and an integer k, return the k most frequent strings.

Return the answer sorted by the frequency from highest to lowest. Sort the words with the same frequency by
↪ their lexicographical order.

Example 1:

```
Input: words = ["i","love","leetcode","i","love","coding"], k = 2
Output: ["i","love"]
Explanation: "i" and "love" are the two most frequent words.
Note that "i" comes before "love" due to a lower alphabetical order.
```
Example 2:
```
Input: words = ["the","day","is","sunny","the","the","the","sunny","is","is"], k = 4
Output: ["the","is","sunny","day"]
Explanation: "the", "is", "sunny" and "day" are the four most frequent words, with the number of occurrence being 4, 3, 2 and 1 respectively.
```

>Solution
```python
class Solution:
    def topKFrequent(self, words: List[str], k: int) -> List[str]:

        # heap 很显然的解法，但不想用这么多额外的开销
        counts = defaultdict(lambda: 0)
        for word in words:
            counts[word] += 1

        inverse = defaultdict(lambda: [])
        for word, count in counts.items():
            heappush(inverse[count], word)

        res = []

        for count in nlargest(k, counts.values()):
            res.append(heappop(inverse[count]))

        return res
```


Example: 658. Find K Closest Elements
https://leetcode.com/problems/find-k-closest-elements/

Given a sorted integer array arr, two integers k and x, return the k closest integers to x in the array. The result should also be sorted in ascending order.

An integer a is closer to x than an integer b if:
```
|a - x| < |b - x|, or
|a - x| == |b - x| and a < b
 ```

Example 1:
```
Input: arr = [1,2,3,4,5], k = 4, x = 3
```

```
Output: [1,2,3,4]
```
Example 2:
```
Input: arr = [1,2,3,4,5], k = 4, x = -1
Output: [1,2,3,4]
```

>Solution
```python
class Solution:
    def findClosestElements(self, arr: List[int], k: int, x: int) -> List[int]:
        # way 1 - double pointer
        # way 2 - binary search

        # way 3 - heap
        import heapq
        heap = []

        for num in arr:
            dis = abs(num - x)
            heapq.heappush(heap, (dis, num))
            # if len(heap) > k: # 用最大堆有问题，就是会忽视这个条件 |a - x| == |b - x| and a < b
            #     heapq.heappop(heap)

        res = []
        i = 0
        while i < k:
            _, num = heapq.heappop(heap)
            res.append(num)
            i += 1

        return sorted(res) # 别忘了最后要 sorted()  如果是 res.sort() 会返回 []

```


632. Smallest Range Covering Elements from K Lists
↪ https://leetcode.com/problems/smallest-range-covering-elements-from-k-lists/
You have k lists of sorted integers in non-decreasing order. Find the smallest range that includes at least one
↪ number from each of the k lists. We define the range [a, b] is smaller than range [c, d] if b - a < d - c or
↪ a < c if b - a == d - c.
```
Example 1:

Input: nums = [[4,10,15,24,26],[0,9,12,20],[5,18,22,30]]
Output: [20,24]
Explanation:
List 1: [4, 10, 15, 24,26], 24 is in range [20,24].
List 2: [0, 9, 12, 20], 20 is in range [20,24].
List 3: [5, 18, 22, 30], 22 is in range [20,24].
Example 2:
```

```
Input: nums = [[1,2,3],[1,2,3],[1,2,3]]
Output: [1,1]
```

```python
class Solution:
    def smallestRange(self, nums: List[List[int]]) -> List[int]:
        # 各种暴力都超时
        # 还有可以 hash + sliding window 就是超级麻烦, double pointer 也是超时

        # 这个是 heap 的解
        rangeLeft, rangeRight = -10**9, 10**9
        maxValue = max(vec[0] for vec in nums)
        priorityQueue = [(vec[0], i, 0) for i, vec in enumerate(nums)]
        heapq.heapify(priorityQueue)

        while True:
            minValue, row, idx = heapq.heappop(priorityQueue)
            if maxValue - minValue < rangeRight - rangeLeft:
                rangeLeft, rangeRight = minValue, maxValue
            if idx == len(nums[row]) - 1:
                break
            maxValue = max(maxValue, nums[row][idx + 1])
            heapq.heappush(priorityQueue, (nums[row][idx + 1], row, idx + 1))

        return [rangeLeft, rangeRight]
```

---

10 DP

```
# DP


□□□ Subarray DP □□□

### 152. Maximum Product Subarray https://leetcode.com/problems/maximum-product-subarray/
Given an integer array nums, find a contiguous non-empty subarray within the array that has the largest product,
↪  and return the product. The test cases are generated so that the answer will fit in a 32-bit integer. A
↪  subarray is a contiguous subsequence of the array.
```
```
Example 1:

Input: nums = [2,3,-2,4]
Output: 6
Explanation: [2,3] has the largest product 6.
Example 2:

Input: nums = [-2,0,-1]
Output: 0
```

Explanation: The result cannot be 2, because [-2,-1] is not a subarray.
```
> 经典 DP 在 subarray 类型的！ olk
```python
class Solution:
    def maxProduct(self, nums: List[int]) -> int:

        # dp 解法和之前不一样的
        n = len(nums)
        dpmax = [0] * n
        dpmin = [0] * n
        dpmax[0] = nums[0]
        dpmin[0] = nums[0]
        res = nums[0]

        for i in range(1, n):
            dpmax[i] = max(dpmax[i-1] * nums[i], nums[i], dpmin[i-1]*nums[i])
            dpmin[i] = min(dpmax[i-1] * nums[i], nums[i], dpmin[i-1]*nums[i])
            res = max(res, dpmax[i])

        return res
```

53. Maximum Subarray
Given an integer array nums, find the contiguous subarray (containing at least one number) which has the largest sum and return its sum. A subarray is a contiguous part of an array.
```
Example 1:

Input: nums = [-2,1,-3,4,-1,2,1,-5,4]
Output: 6
Explanation: [4,-1,2,1] has the largest sum = 6.
Example 2:

Input: nums = [1]
Output: 1
Example 3:

Input: nums = [5,4,-1,7,8]
Output: 23
```
> subarray 很多都是 DP 的，有一部分可以 double pointer
```python
class Solution:
    def maxSubArray(self, nums: List[int]) -> int:
        # dp[i] 数组的含义是以 nums[i] 结尾的连续子数组的最大和
        # 关键是 dp[i] 转换的含义，这里面要判断 dp[i-1] 的大小，如果 >0, dp[i] 会更大，但是 dp[i-1]<0, 重新开始，因为 nums[i] 会更小如果加上 dp[i-1]

        n = len(nums)
        dp = [0] * n
        dp[0] = nums[0]

```
        for i in range(1, n):
            if dp[i-1] > 0: # 不是判断 nums[i] 的正负
                dp[i] = dp[i-1] + nums[i]
            else:
                dp[i] = nums[i]
        return max(dp) # 返回所有的最大值

        # brute force - 这是比较优化的方法，但超时
        res = -inf
        n = len(nums)
        for i in range(n):
            curr = 0
            for j in range(i, n):
                curr += nums[j]
                res = max(res, curr)

        return res
```


718. Maximum Length of Repeated Subarray
Given two integer arrays nums1 and nums2, return the maximum length of a subarray that appears in both arrays.
```
Example 1:

Input: nums1 = [1,2,3,2,1], nums2 = [3,2,1,4,7]
Output: 3
Explanation: The repeated subarray with maximum length is [3,2,1].
Example 2:

Input: nums1 = [0,0,0,0,0], nums2 = [0,0,0,0,0]
Output: 5
```

```python
class Solution:
    def findLength(self, nums1: List[int], nums2: List[int]) -> int:

        # #  动态规划可以，和之前的 maximum subsequence 一样的 dp 方程
        # A = nums1
        # B = nums2
        # n, m = len(A), len(B)
        # dp = [[0] * (m + 1) for _ in range(n + 1)]
        # ans = 0
        # for i in range(n - 1, -1, -1):
        #     for j in range(m - 1, -1, -1):
        #         dp[i][j] = dp[i + 1][j + 1] + 1 if A[i] == B[j] else 0
        #         ans = max(ans, dp[i][j])
        # return ans

        # time o(n x m) and space o(n x m)

```

```
        # way 2 - sliding window
        def maxLength(addA: int, addB: int, length: int) -> int:
            ret = k = 0
            for i in range(length):
                if A[addA + i] == B[addB + i]:
                    k += 1
                    ret = max(ret, k)
                else:
                    k = 0
            return ret

        A = nums1
        B = nums2
        n, m = len(A), len(B)
        ret = 0
        for i in range(n):
            length = min(m, n - i)
            ret = max(ret, maxLength(i, 0, length))
        for i in range(m):
            length = min(n, m - i)
            ret = max(ret, maxLength(0, i, length))
        return ret

# 作者: LeetCode-Solution
# 链接:
↪ https://leetcode-cn.com/problems/maximum-length-of-repeated-subarray/solution/zui-chang-zhong-fu-zi-shu-zu-by-leetcod
# 来源: 力扣 (LeetCode)
# 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
```

Stock price

2110. Number of Smooth Descent Periods of a Stock

You are given an integer array prices representing the daily price history of a stock, where prices[i] is the
↪ stock price on the ith day. A smooth descent period of a stock consists of one or more contiguous days such
↪ that the price on each day is lower than the price on the preceding day by exactly 1. The first day of the
↪ period is exempted from this rule. Return the number of smooth descent periods.

```
Example 1:

Input: prices = [3,2,1,4]
Output: 7
Explanation: There are 7 smooth descent periods:
[3], [2], [1], [4], [3,2], [2,1], and [3,2,1]
Note that a period with one day is a smooth descent period by the definition.
Example 2:
```

```
Input: prices = [8,6,7,7]
Output: 4
Explanation: There are 4 smooth descent periods: [8], [6], [7], and [7]
Note that [8,6] is not a smooth descent period as 8 - 6 ≠ 1.
Example 3:

Input: prices = [1]
Output: 1
Explanation: There is 1 smooth descent period: [1]
```

```python
class Solution:
    def getDescentPeriods(self, prices: List[int]) -> int:
        n = len(prices)
        res = 1   # 平滑下降阶段的总数，初值为 dp[0]
        prev = 1   # 上一个元素为结尾的平滑下降阶段的总数，初值为 dp[0]
        # 从 1 开始遍历数组，按照递推式更新 prev 以及总数 res
        for i in range(1, n):
            if prices[i] == prices[i-1] - 1:
                prev += 1
            else:
                prev = 1
            res += prev
        return res

    # time o(n) and space o(1) 算是非常简单的 dp 了，需要理解 dp 的定义和含义!
```

---

11 Linked list

12 Sorting

13 Prefix sum

```
# Prefix sum

## 很多是 subarray 的题

### 238. Product of Array Except Self https://leetcode.com/problems/product-of-array-except-self/

Given an integer array nums, return an array answer such that answer[i] is equal to the product of all the
↪  elements of nums except nums[i]. The product of any prefix or suffix of nums is guaranteed to fit in a
↪  32-bit integer. You must write an algorithm that runs in O(n) time and without using the division operation.
```
Example 1:

Input: nums = [1,2,3,4]
Output: [24,12,8,6]

Example 2:

Input: nums = [-1,1,0,-3,3]
Output: [0,0,9,0,0]
```

```python
class Solution:
    def productExceptSelf(self, nums: List[int]) -> List[int]:
        n = len(nums)
        ans = [0]*n

        ans[0] = 1
        for i in range(1, n):
            ans[i] = ans[i-1]*nums[i-1]

        R = 1
        for i in reversed(range(n)):
            ans[i] = ans[i] * R
            R *= nums[i]

        return ans
```

325. Maximum Size Subarray Sum Equals k https://leetcode.com/problems/maximum-size-subarray-sum-equals-k/

Given an integer array nums and an integer k, return the maximum length of a subarray that sums to k. If there is not one, return 0 instead.

```
Example 1:

Input: nums = [1,-1,5,-2,3], k = 3
Output: 4
Explanation: The subarray [1, -1, 5, -2] sums to 3 and is the longest.
Example 2:

Input: nums = [-2,-1,2,1], k = 1
Output: 2
Explanation: The subarray [-1, 2] sums to 1 and is the longest.
```

```python
class Solution:
    def maxSubArrayLen(self, nums: List[int], k: int) -> int:
        # presum 的题
        hashmap = {0:-1}
        pre_sum = 0
        res = 0
        for i, num in enumerate(nums):
            pre_sum += num
            # Check if all of the numbers seen so far sum to k. 非常有必要! 否则容易错!
            if pre_sum == k:
```
```

```
                res = i + 1
            # If any subarray seen so far sums to k, then
            # update the length of the longest_subarray.
            if pre_sum - k in hashmap:
                res = max(res, i - hashmap[pre_sum - k])
            # Only add the current prefix_sum index pair to the
            # map if the prefix_sum is not already in the map.
            if pre_sum not in hashmap: # 这个判断有必要，因为可能正负都有，之前的题没有判断这个因为递增
                hashmap[pre_sum] = hashmap.get(pre_sum, 0) + i

        return res

    # time o(n), space o(n)
```

560. Subarray Sum Equals K

Given an array of integers nums and an integer k, return the total number of subarrays whose sum equals to k. A subarray is a contiguous non-empty sequence of elements within an array.

```
Example 1:

Input: nums = [1,1,1], k = 2
Output: 2
Example 2:

Input: nums = [1,2,3], k = 3
Output: 2
```

```python
class Solution:
    def subarraySum(self, nums: List[int], k: int) -> int:

#
https://leetcode-cn.com/problems/subarray-sum-equals-k/solution/qian-zhui-he-si-xiang-560-he-wei-kde-zi-shu-zu-by-/

        # 很高频的，前缀和的几种写法
        # # way1 - brute force 超时，o(n^2), space o(1)
        # res = 0
        # for i in range(len(nums)):
        #     for j in range(i, len(nums)):
        #         if sum(nums[i:j+1]) == k:
        #             res += 1
        # return res


        # prefix sum - 也是超时了  o(n^2), space o (n) 因为有 pre 的空间
#         cnt, n =  0, len(nums)
#         pre = [0] * (n + 1)
#         for i in range(1, n + 1):
#             pre[i] = pre[i - 1] + nums[i - 1]

```

```
#         for i in range(1, n + 1):
#             for j in range(i, n + 1):
#                 if (pre[j] - pre[i - 1] == k): cnt += 1
#         return cnt

        # hashmap + pre_sum 这个写法更适合我之前的习惯
        pre, res = 0, 0
        count = dict()
        for num in nums:
            pre += num
            if pre == k: res += 1
            res += count.get(pre-k, 0)
            count[pre] = count.get(pre, 0) + 1 # 这个写法更好一些，和 sliding window 一样

        return res


#         pre_sum = collections.defaultdict(int)
#         res, cur_pre_sum = 0, 0
#         for i in range(len(nums)):
#             cur_pre_sum += nums[i]
#             if cur_pre_sum - k in pre_sum:
#                 res += pre_sum[cur_pre_sum]
#             pre_sum[cur_pre_sum] += 1
#         return res


        # # 记录 绿色 " 前缀和" (从 0 到 i 的前缀和) 的 值和出现的次数.
        # pre_sum = collections.defaultdict(int)
        # # 初始化 前缀和 为 0 的 子序列 出现了 一次.
        # # 对应 第一类情况，上面的 if cur_pre_sum - k == 0 语句
        # pre_sum[0] = 1
        # # 记录 当前 位置的 前缀和
        # cur_pre_sum = 0
        # # 用于记录结果
        # res = 0
        # for i in range(len(nums)):
        #     cur_pre_sum += nums[i]  # 计算 当前位置的 前缀和
        #     # cur_sum - k 是我们想找的前缀和 nums[0..j]
        #     # 如果前面有这个前缀和，则直接更新答案
        #     green_sum = cur_pre_sum - k
        #     if green_sum in pre_sum:
        #         res += pre_sum[green_sum]
        #     # 每次计算都将前缀和加入字典
        #     pre_sum[cur_pre_sum] += 1
        # return res
```

974. Subarray Sums Divisible by K

Given an integer array nums and an integer k, return the number of non-empty subarrays that have a sum divisible by k. A subarray is a contiguous part of an array.

```
```

````
Example 1:

Input: nums = [4,5,0,-2,-3,1], k = 5
Output: 7
Explanation: There are 7 subarrays with a sum divisible by k = 5:
[4, 5, 0, -2, -3, 1], [5], [5, 0], [5, 0, -2, -3], [0], [0, -2, -3], [-2, -3]
Example 2:

Input: nums = [5], k = 9
Output: 0
```
```python
class Solution:
    def subarraysDivByK(self, nums: List[int], k: int) -> int:
        # res, cnt = 0, Counter({0:1})    # 定义哨兵节点，取余结果为 0 时，默认已经出现一次
        res, cnt = 0, {0:1}    # 改成 {} 不能直接调用，会报错，如果之前没有出现
        pre = list(accumulate(nums, add))    # 计算前缀和数组

        for i in range(len(pre)):    # 遍历每个前缀和元素
            mod = (pre[i]+k) % k    # 因为可能存在负数，所以加上一个 k，再计算对 k 取余结果
            res += cnt[mod]    # 加上哈希表中存储的 mod 对应的次数，更新可行方案数，会报错
            cnt[mod] += 1    # 更新哈希表
            print('mod',mod)
            print('cnt',cnt)
            print('res',res)

        return res

#         # 这个题挺好的，不需要做判断，还是同余的道理，
#         pre_sum, res = 0, 0
#         hashmap = {} # 这样初始化，然后把 prefix sum 求和的情况在 for loop 里面判断

#         for i, num in enumerate(nums):
#             pre_sum += num
#             if pre_sum % k == 0:
#                 res += 1
#             reminder = pre_sum % k
#             res += hashmap.get(reminder,0)
#             hashmap[reminder] = hashmap.get(reminder, 0) + 1

#             # 很多时候会考虑到判断，但是少一个解？
#             # if reminder in hashmap:
#             #     res += hashmap[reminder]
#             # else:
#             #     hashmap[reminder] = hashmap.get(reminder, 0) + 1

#         return res
```

### 1590. Make Sum Divisible by P
````
````

Given an array of positive integers nums, remove the smallest subarray (possibly empty) such that the sum of the remaining elements is divisible by p. It is not allowed to remove the whole array. Return the length of the smallest subarray that you need to remove, or -1 if it's impossible. A subarray is defined as a contiguous block of elements in the array.

```
Example 1:

Input: nums = [3,1,4,2], p = 6
Output: 1
Explanation: The sum of the elements in nums is 10, which is not divisible by 6. We can remove the subarray [4], and the sum of the remaining elements is 6, which is divisible by 6.
Example 2:

Input: nums = [6,3,5,2], p = 9
Output: 2
Explanation: We cannot remove a single element to get a sum divisible by 9. The best way is to remove the subarray [5,2], leaving us with [6,3] with sum 9.
Example 3:

Input: nums = [1,2,3], p = 3
Output: 0
Explanation: Here the sum is 6. which is already divisible by 3. Thus we do not need to remove anything.
```

```python
class Solution:
    def minSubarray(self, nums: List[int], p: int) -> int:

        pre = list(accumulate(nums, add))
        mod = pre[-1] % p
        hashT = {0:-1}
        if mod == 0: return 0

        res = len(nums)
        for i in range(len(nums)):
            curmod = pre[i] % p
            tarmod = (curmod - mod + p) % p
            if tarmod in hashT:
                dis = i - hashT[tarmod]
                res = dis if dis < res else res
                if res == 1 and len(nums) != 1:
                    return 1
            hashT[curmod] = i
        if res == len(nums):
            res = -1
        return res
```

---

14 Array/String

15 Math

```
# Math 相关题型


## Randomized 随机操作

这个题以及相关题目有很多可以考的
1. 2个 uniform distribution 求和不是 uniform，因为有很多重复的可能性，比如 rand2() + rand2() 有多种可能性得到 3，这个问题也可以用数学公式推导，他的分布是三角分布
2. rand_x() 生成 [1,x] 那么 (rand_x - 1) * Y + rand_y() 可以生成 rand_{xy} [1, xy] 的随机数 - 这个定理很重要
3. rejected sampling，生成之后可以用 mod 操作来拒绝掉不想要的部分，但注意效率，可以迭代几次来提高效率
4. 扔骰子问题，或者抛硬币
   https://leetcode.cn/problems/implement-rand10-using-rand7/solution/wei-rao-li-lun-yi-ge-bu-jun-yun-ying-bi-fo4ei/
5. 只有 2 个 Gaussian 求和是 Gaussian. 2个 Uniform 的加减乘除都不是 uniform
6. 2个均匀分布 求和是三角分布，这个知乎讲得挺好的!   https://www.zhihu.com/question/27060339


### 470. Implement Rand10() Using Rand7() https://leetcode.com/problems/implement-rand10-using-rand7/

Given the API rand7() that generates a uniform random integer in the range [1, 7], write a function rand10() that generates a uniform random integer in the range [1, 10]. You can only call the API rand7(), and you shouldn't call any other API. Please do not use a language's built-in random API.

Each test case will have one internal argument n, the number of times that your implemented function rand10() will be called while testing. Note that this is not an argument passed to rand10().

Example 1:
```
Input: n = 1
Output: [2]
```
Example 2:
```
Input: n = 2
Output: [2,8]
```
Example 3:
```
Input: n = 3
Output: [3,8,10]
```
Constraints:1 <= n <= 105
Follow up: What is the expected value for the number of calls to rand7() function? Could you minimize the number of calls to rand7()?


```python
class Solution:
```
```

```
    def rand10(self):
        """
        :rtype: int
        """
        while True:
            base = rand7() - 1
            new = (rand7() - 1) * 7 + rand7() # 49
            if new <= 40:
                return new % 10 + 1


class Solution: # 解决 follow-up 的 question!
    def rand10(self) -> int:
        while True:
            a = rand7()
            b = rand7()
            idx = (a - 1) * 7 + b
            if idx <= 40:
                return 1 + (idx - 1) % 10 # 拒绝了 41-49 9 个数
            a = idx - 40
            b = rand7()
            # get uniform dist from 1 - 63 - 拒绝 61,62,63 三个数
            idx = (a - 1) * 7 + b
            if idx <= 60:
                return 1 + (idx - 1) % 10
            a = idx - 60
            b = rand7()
            # get uniform dist from 1 - 21 - 这样每次只拒绝 21 一个数
            idx = (a - 1) * 7 + b
            if idx <= 20:
                return 1 + (idx - 1) % 10
```


进位制理解和操作 - digit
基本操作不熟悉!
```python
# 就是对进位不了解或者不熟悉
num = 123

print(num // 10)  # 最高位 + 次高位
print(num // 100) # 最高位
print(num // 1000) # 0

print(num % 10) # 个位
print(num % 100) # 十位 + 个位
print(num % 1000) # 百分位 + 十位 + 个位

# output
# 12
# 1
```

```
# 0
# 3
# 23
# 123
````

400. Nth Digit https://leetcode.com/problems/nth-digit/

Given an integer n, return the nth digit of the infinite integer sequence [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ...].

Example 1:
```
Input: n = 3
Output: 3
```
Example 2:
```
Input: n = 11
Output: 0
Explanation: The 11th digit of the sequence 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ... is a 0, which is part of the number 10.
```

``` python
class Solution:
    def totalDigits(self, length: int) -> int:
        digits = 0
        curCount = 9
        for curLength in range(1, length + 1):
            digits += curLength * curCount
            curCount *= 10
        return digits

    def findNthDigit(self, n: int) -> int:
        low, high = 1, 9
        while low < high:
            mid = (low + high) // 2
            if self.totalDigits(mid) < n:
                low = mid + 1
            else:
                high = mid
        d = low # 返回属于的位数

        prevDigits = self.totalDigits(d - 1) # 之前的所有数 digit 求和
        index = n - prevDigits - 1 # 当前位数的 index，比如 2 位数中排序
        start = 10 ** (d - 1) # 之前有多少个数，绝对 number，比,1，2，3，...，100
        num = start + index // d   # n 对应的当前的数
        digitIndex = index % d # 判断是 d 位数中第几个，百分位还是十分位，个位
        print(d, prevDigits,index,start, num, digitIndex)
        print(num // 10 ** (d - digitIndex - 1)) # 这是关键，返回一个 digit，都要%10
        return num // 10 ** (d - digitIndex - 1) % 10
```

```
        # example 121
        # print 2 9 111 10 65 1
        # print 65

class Solution:
    def findNthDigit(self, n: int) -> int:
        cur, base = 1, 9
        while n > cur * base:
            n -= cur * base
            cur += 1
            base *= 10
        print(cur,n)
        n -= 1
        # 数字
        num = 10 ** (cur - 1) + n // cur
        print(num)
        # 数字里的第几位
        idx = n % cur
        print(idx)
        return num // (10 ** (cur - 1 - idx)) % 10
```

---

16 Special topics

16.0.1 Subarray

很多 prefix sum 还有 dp

16.0.2 Substring

多数是 sliding window

16.0.3 Subsequence

17 Probability and Statistics

17.1 Probability

17.1.0.1 Sum of two uniform distribution ? 这个帖子讲了 2 个 u(0,1) 的加减乘除的方法, 只有 Gaussian 求和是 Gaussian 其他基本都不是.

参考文献