滑动窗口

209. 长度最小的子数组

给定一个含有 `n` 个正整数的数组和一个正整数 `target` 。

找出该数组中满足其总和大于等于 `target` 的长度最小的 **连续**

子数组

`[numsl, numsl+1, ..., numsr-1, numsr]` ，并返回其长度。如果不存在符合条件的子数组，返回 `0` 。

示例 1：

```
输入：target = 7, nums = [2,3,1,2,4,3]
输出：2
解释：子数组 [4,3] 是该条件下的长度最小的子数组。
```

示例 2：

```
输入：target = 4, nums = [1,4,4]
输出：1
```

示例 3：

```
输入：target = 11, nums = [1,1,1,1,1,1,1,1]
输出：0
```

提示：

- `1 <= target <= 109`
- `1 <= nums.length <= 105`
- `1 <= nums[i] <= 105`

1. Java

```java
class Solution {
    public int minSubArrayLen(int target, int[] nums) {
        int n = nums.length;
        int ans = n + 1;
        int s = 0;
        int left = 0;
        for (int right = 0; right < n; right++) {
            s += nums[right];
            while (s - nums[left] >= target) {
                s -= nums[left++];
            }
            if (s >= target) {
```

```
                ans = Math.min(ans, right - left + 1);
            }
        }
        return ans <= n ? ans : 0;
    }
}
```

2. Python

```python
class Solution:
    def minSubArrayLen(self, target: int, nums: List[int]) -> int:
        n = len(nums)
        ans = n + 1
        left = 0
        s = 0
        for right, x in enumerate(nums):
            s += x
            while s - nums[left] >= target:
                s -= nums[left]
                left += 1
            if s >= target:
                ans = min(ans, right - left + 1)
        return ans if ans <= n else 0
```

3. 无重复字符的最长子串

给定一个字符串 `s` ，请你找出其中不含有重复字符的 **最长**

子串

的长度。

示例 1:

```
输入: s = "abcabcbb"
输出: 3
解释: 因为无重复字符的最长子串是 "abc"，所以其长度为 3。
```

示例 2:

```
输入: s = "bbbbb"
输出: 1
解释: 因为无重复字符的最长子串是 "b"，所以其长度为 1。
```

示例 3:

```
输入: s = "pwwkew"
输出: 3
解释: 因为无重复字符的最长子串是 "wke"，所以其长度为 3。
     请注意，你的答案必须是 子串 的长度，"pwke" 是一个子序列，不是子串。
```

提示：

- `0 <= s.length <= 5 * 104`
- `s` 由英文字母、数字、符号和空格组成

1. Java

```java
class Solution {
    public int lengthOfLongestSubstring(String S) {
        char[] s = S.toCharArray();
        int n = s.length;
        int left = 0;
        int ans = 0;
        // 这里的 boolean 同样可以用 HashSet 来实现
        boolean[] has = new boolean[128];
        for (int right = 0; right < n; right++) {
            char c = s[right];
            while (has[c]) {
                has[s[left++]] = false;
            }
            has[c] = true;
            ans = Math.max(ans, right - left + 1);
        }
        return ans;
    }
}
```

2. Python

```python
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        ans = 0
        cnt = Counter()
        left = 0
        for right, x in enumerate(s):
            cnt[x] += 1
            while cnt[x] > 1:
                cnt[s[left]] -= 1
                left += 1
            ans = max(ans, right - left + 1)
        return ans
```

713. 乘积小于 K 的子数组

给你一个整数数组 `nums` 和一个整数 `k` ，请你返回子数组内所有元素的乘积严格小于 `k` 的连续子数组的数目。

示例 1：

```
输入：nums = [10,5,2,6], k = 100
输出：8
解释：8 个乘积小于 100 的子数组分别为：[10]、[5]、[2],、[6]、[10,5]、[5,2]、[2,6]、[5,2,6]。
需要注意的是 [10,5,2] 并不是乘积小于 100 的子数组。
```

示例 2：

```
输入：nums = [1,2,3], k = 0
输出：0
```

提示：

- `1 <= nums.length <= 3 * 104`
- `1 <= nums[i] <= 1000`
- `0 <= k <= 106`

1. Java

```java
class Solution {
    public int numSubarrayProductLessThanK(int[] nums, int k) {
        if (k <= 1) {
            return 0;
        }
        int ans = 0;
        int left = 0;
        int prod = 1;
        for (int right = 0; right < nums.length; right++) {
            prod *= nums[right];
            while (prod >= k) {
                prod /= nums[left++];
            }
            ans += right - left + 1;
        }
        return ans;
    }
}
```

2. Python

```python
class Solution:
    def numSubarrayProductLessThanK(self, nums: List[int], k: int) -> int:
        if k <= 1:
            return 0
        ans = 0
        left = 0
        prod = 1
        for right, x in enumerate(nums):
            prod *= x
            while prod >= k:
                prod /= nums[left]
                left += 1
            ans += right - left + 1
        return ans
```

2958. 最多 K 个重复元素的最长子数组

给你一个整数数组 `nums` 和一个整数 `k` 。

一个元素 `x` 在数组中的 **频率** 指的是它在数组中的出现次数。

如果一个数组中所有元素的频率都 **小于等于** `k` ，那么我们称这个数组是 **好** 数组。

请你返回 `nums` 中 **最长好** 子数组的长度。

子数组 指的是一个数组中一段连续非空的元素序列。

示例 1：

```
输入：nums = [1,2,3,1,2,3,1,2], k = 2
输出：6
解释：最长好子数组是 [1,2,3,1,2,3] ，值 1 ，2 和 3 在子数组中的频率都没有超过 k = 2 。
[2,3,1,2,3,1] 和 [3,1,2,3,1,2] 也是好子数组。
最长好子数组的长度为 6 。
```

示例 2：

```
输入：nums = [1,2,1,2,1,2,1,2], k = 1
输出：2
解释：最长好子数组是 [1,2] ，值 1 和 2 在子数组中的频率都没有超过 k = 1 。[2,1] 也是好子数组。
最长好子数组的长度为 2 。
```

示例 3：

```
输入：nums = [5,5,5,5,5,5,5], k = 4
输出：4
解释：最长好子数组是 [5,5,5,5] ，值 5 在子数组中的频率没有超过 k = 4 。
最长好子数组的长度为 4 。
```

提示：

- `1 <= nums.length <= 105`
- `1 <= nums[i] <= 109`
- `1 <= k <= nums.length`

1. Java

```java
class Solution {
    public int maxSubarrayLength(int[] nums, int k) {
        int ans = 0;
        int left = 0;
        HashMap<Integer, Integer> cnt = new HashMap<>();
        for (int right = 0; right < nums.length; right++) {
            cnt.merge(nums[right], 1, Integer::sum);
            while (cnt.get(nums[right]) > k) {
                cnt.merge(nums[left], -1, Integer::sum);
                left++;
            }
            ans = Math.max(ans, right - left + 1);
        }
        return ans;
    }
```

```
}
```

2. Python

```python
class Solution:
    def maxSubarrayLength(self, nums: List[int], k: int) -> int:
        ans = 0
        cnt = Counter()
        left = 0
        for right, x in enumerate(nums):
            cnt[x] += 1
            while cnt[x] > k:
                cnt[nums[left]] -= 1
                left += 1
            ans = max(ans, right - left + 1)
        return ans
```

2730. 找到最长的半重复子字符串

给你一个下标从 **0** 开始的字符串 `s` ，这个字符串只包含 `0` 到 `9` 的数字字符。

如果一个字符串 `t` 中至多有一对相邻字符是相等的，那么称这个字符串 `t` 是 **半重复的** 。例如， `0010` 、 `002020` 、 `0123` 、 `2002` 和 `54944` 是半重复字符串，而 `00101022` 和 `1101234883` 不是。

请你返回 `s` 中最长 **半重复** 子字符串的长度。

一个 **子字符串** 是一个字符串中一段连续 **非空** 的字符。

示例 1：

```
输入：s = "52233"
输出：4
解释：最长半重复子字符串是 "5223" ，子字符串从 i = 0 开始，在 j = 3 处结束。
```

示例 2：

```
输入：s = "5494"
输出：4
解释：s 就是一个半重复字符串，所以答案为 4 。
```

示例 3：

```
输入：s = "1111111"
输出：2
解释：最长半重复子字符串是 "11" ，子字符串从 i = 0 开始，在 j = 1 处结束。
```

提示：

- `1 <= s.length <= 50`
- `'0' <= s[i] <= '9'`

1. Java

```java
class Solution {
    public int longestSemiRepetitiveSubstring(String S) {
        int left = 0;
        int ans = 1;
        int sameCnt = 0;
        char[] s = S.toCharArray();
        for (int right = 1; right < s.length; right++) {
            if (s[right] == s[right - 1]) {
                sameCnt++;
            }
            if (sameCnt > 1) {
                left++;
                while (s[left] != s[left - 1]) {
                    left++;
                }
                sameCnt = 1;
            }
            ans = Math.max(ans, right - left + 1);
        }
        return ans;
    }
}
```

2. Python

```python
class Solution:
    def longestSemiRepetitiveSubstring(self, s: str) -> int:
        # 0 也是半重复子字符串
        ans = 1
        left = 0
        same_cnt = 0
        for right in range(1, len(s)):
            if s[right] == s[right - 1]:
                same_cnt += 1
            if same_cnt > 1:
                left += 1
                while s[left] != s[left - 1]:
                    left += 1
                same_cnt = 1
            ans = max(ans, right - left + 1)
        return ans
```

1004. 最大连续1的个数 III

给定一个二进制数组 `nums` 和一个整数 `k`，如果可以翻转最多 `k` 个 `0` ，则返回 数组中连续 `1` 的最大个数 。

示例 1：

```
输入：nums = [1,1,1,0,0,0,1,1,1,1,0], K = 2
输出：6
解释：[1,1,1,0,0,1,1,1,1,1,1]
粗体数字从 0 翻转到 1，最长的子数组长度为 6。
```

示例 2：

```
输入：nums = [0,0,1,1,0,0,1,1,1,0,1,1,0,0,0,1,1,1,1], K = 3
输出：10
解释：[0,0,1,1,1,1,1,1,1,1,1,1,0,0,0,1,1,1,1]
粗体数字从 0 翻转到 1，最长的子数组长度为 10。
```

提示：

- `1 <= nums.length <= 105`
- `nums[i]` 不是 `0` 就是 `1`
- `0 <= k <= nums.length`

1. Java

```java
class Solution {
    public int longestOnes(int[] nums, int k) {
        int ans = 0;
        int left = 0;
        int cnt0 = 0;
        for (int right = 0; right < nums.length; right++) {
            cnt0 += 1 - nums[right];
            while (cnt0 > k) {
                cnt0 -= 1 - nums[left++];
            }
            ans = Math.max(ans, right - left + 1);
        }
        return ans;
    }
}
```

2. Python

```python
class Solution:
    def longestOnes(self, nums: List[int], k: int) -> int:
        ans = 0
        left = 0
        cnt0 = 0
        for right, x in enumerate(nums):
            cnt0 += 1 - x
            while cnt0 > k:
                cnt0 -= 1 - nums[left]
                left += 1
            ans = max(ans, right - left + 1)
        return ans
```

2962. 统计最大元素出现至少 K 次的子数组

给你一个整数数组 `nums` 和一个 **正整数** `k` 。

请你统计有多少满足 「`nums` 中的 **最大** 元素」至少出现 `k` 次的子数组，并返回满足这一条件的子数组的数目。

子数组是数组中的一个连续元素序列。

示例 1：

```
输入：nums = [1,3,2,3,3], k = 2
输出：6
解释：包含元素 3 至少 2 次的子数组为：[1,3,2,3]、[1,3,2,3,3]、[3,2,3]、[3,2,3,3]、[2,3,3] 和 [3,3] 。
```

示例 2：

```
输入：nums = [1,4,2,1], k = 3
输出：0
解释：没有子数组包含元素 4 至少 3 次。
```

提示：

- `1 <= nums.length <= 105`
- `1 <= nums[i] <= 106`
- `1 <= k <= 105`

1. Java

```java
class Solution {
    // 注意这里是 long
    public long countSubarrays(int[] nums, int k) {
        int left = 0;
        long ans = 0L;
        int a = Arrays.stream(nums).max().getAsInt();
        int cnt = 0;
        for (int right = 0; right < nums.length; right++) {
            if (nums[right] == a) {
                cnt++;
            }
            while (cnt == k) {
                if (nums[left] == a) {
                    cnt--;
                }
                left++;
            }
            ans += left;
        }
        return ans;
    }
}
```

2. Python

```python
class Solution:
    def countSubarrays(self, nums: List[int], k: int) -> int:
        left = 0
        ans = 0
        a = max(nums)
        cnt = 0
        for right, x in enumerate(nums):
            if x == a:
                cnt += 1
            while cnt == k:
                if nums[left] == a:
                    cnt -= 1
                left += 1
            ans += left
        return ans
```

2302. 统计得分小于 K 的子数组数目

一个数组的 **分数** 定义为数组之和 **乘以** 数组的长度。

- 比方说，`[1, 2, 3, 4, 5]` 的分数为 `(1 + 2 + 3 + 4 + 5) * 5 = 75` 。

给你一个正整数数组 `nums` 和一个整数 `k` ，请你返回 `nums` 中分数 **严格小于** `k` 的 **非空整数子数组数目**。

子数组 是数组中的一个连续元素序列。

示例 1：

```
输入：nums = [2,1,4,3,5], k = 10
输出：6
解释：
有 6 个子数组的分数小于 10 ：
- [2] 分数为 2 * 1 = 2 。
- [1] 分数为 1 * 1 = 1 。
- [4] 分数为 4 * 1 = 4 。
- [3] 分数为 3 * 1 = 3 。
- [5] 分数为 5 * 1 = 5 。
- [2,1] 分数为 (2 + 1) * 2 = 6 。
注意，子数组 [1,4] 和 [4,3,5] 不符合要求，因为它们的分数分别为 10 和 36，但我们要求子数组的分数严格小于 10 。
```

示例 2：

```
输入：nums = [1,1,1], k = 5
输出：5
解释：
除了 [1,1,1] 以外每个子数组分数都小于 5 。
[1,1,1] 分数为 (1 + 1 + 1) * 3 = 9 ，大于 5 。
所以总共有 5 个子数组得分小于 5 。
```

提示：

- `1 <= nums.length <= 105`
- `1 <= nums[i] <= 105`
- `1 <= k <= 1015`

1. Java

```java
class Solution {
    public long countSubarrays(int[] nums, long k) {
        long ans = 0L;
        int left = 0;
        long s = 0L;
        for (int right = 0; right < nums.length; right++) {
            s += nums[right];
            while (s * (right - left + 1) >= k) {
                s -= nums[left++];
            }
            ans += right - left + 1;
        }
        return ans;
    }
}
```

2. Python

```python
class Solution:
    def countSubarrays(self, nums: List[int], k: int) -> int:
        ans = 0
        left = 0
        s = 0
        for right, x in enumerate(nums):
            s += x
            while s * (right - left + 1) >= k:
                s -= nums[left]
                left += 1
            ans += right - left + 1
        return ans
```

1658. 将 x 减到 0 的最小操作数

给你一个整数数组 `nums` 和一个整数 `x` 。每一次操作时，你应当移除数组 `nums` 最左边或最右边的元素，然后从 `x` 中减去该元素的值。请注意，需要 **修改** 数组以供接下来的操作使用。

如果可以将 `x` **恰好** 减到 `0` ，返回 **最小操作数** ；否则，返回 `-1` 。

示例 1：

```
输入：nums = [1,1,4,2,3], x = 5
输出：2
解释：最佳解决方案是移除后两个元素，将 x 减到 0 。
```

示例 2：

```
输入：nums = [5,6,7,8,9], x = 4
输出：-1
```

示例 3：

```
输入：nums = [3,2,20,1,1,3], x = 10
输出：5
解释：最佳解决方案是移除后三个元素和前两个元素（总共 5 次操作），将 x 减到 0 。
```

提示：

- `1 <= nums.length <= 105`
- `1 <= nums[i] <= 104`
- `1 <= x <= 109`

1. Java

```java
class Solution {
    public int minOperations(int[] nums, int x) {
        int target = Arrays.stream(nums).sum() - x;
        if (target < 0) {
            return -1;
        }
        int ans = -1;
        int left = 0;
        int s = 0;
        for (int right = 0; right < nums.length; right++) {
            s += nums[right];
            while (s > target) {
                s -= nums[left++];
            }
            if (s == target) {
                ans = Math.max(ans, right - left + 1);
            }
        }
        return ans < 0 ? -1 : nums.length - ans;
    }
}
```

2. Python

```python
class Solution:
    def minOperations(self, nums: List[int], x: int) -> int:
        target = sum(nums) - x
        if target < 0:
            return -1
        ans = -1
        left = 0
        s = 0
        for right, x in enumerate(nums):
            s += x
            while s > target:
                s -= nums[left]
```

```
                left += 1
            if s == target:
                ans = max(ans, right - left + 1)
        return -1 if ans < 0 else len(nums) - ans
```

76. 最小覆盖子串

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

注意：

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

示例 1：

```
输入：s = "ADOBECODEBANC", t = "ABC"
输出："BANC"
解释：最小覆盖子串 "BANC" 包含来自字符串 t 的 'A'、'B' 和 'C'。
```

示例 2：

```
输入：s = "a", t = "a"
输出："a"
解释：整个字符串 s 是最小覆盖子串。
```

示例 3:

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

提示：

- `m == s.length`
- `n == t.length`
- `1 <= m, n <= 105`
- `s` 和 `t` 由英文字母组成

1. Java

```java
class Solution {
    public String minWindow(String S, String t) {
        char[] s = S.toCharArray();
        int m = s.length;
        int ansLeft = -1;
        int ansRight = m;
        int left = 0;
```

```
        int[] cntS = new int[128];
        int[] cntT = new int[128];
        for (char c : t.toCharArray()) {
            cntT[c]++;
        }
        for (int right = 0; right < m; right++) {
            cntS[s[right]]++;
            while (isCovered(cntS, cntT)) {
                if (right - left < ansRight - ansLeft) {
                    ansLeft = left;
                    ansRight = right;
                }
                cntS[s[left++]]--;
            }
        }
        return ansLeft < 0 ? "" : S.substring(ansLeft, ansRight + 1);
    }

    private boolean isCovered(int[] cntS, int[] cntT) {
        for (int i = 'A'; i <= 'Z'; i++) {
            if (cntS[i] < cntT[i]) {
                return false;
            }
        }
        for (int i = 'a'; i <= 'z'; i++) {
            if (cntS[i] < cntT[i]) {
                return false;
            }
        }
        return true;
    }
}
```

2. Python

```
class Solution:
    def minWindow(self, s: str, t: str) -> str:
        ans_left, ans_right = -1, len(s)
        left = 0
        cnt_s = Counter()
        cnt_t = Counter(t)
        for right, c in enumerate(s):
            cnt_s[c] += 1
            while cnt_s >= cnt_t:
                if right - left < ans_right - ans_left:
                    ans_left, ans_right = left, right
                cnt_s[s[left]] -= 1
                left += 1
        return "" if ans_left < 0 else s[ans_left : ans_right + 1]
```