双指针 one pager

MLE 算法指北

2025 年 2 月 6 日

1 双指针概述

双指针（Two Pointers）是一种常见的算法技巧，适用于数组、字符串、链表等数据结构，主要用于优化时间复杂度，减少不必要的遍历。在 $O(n^2)$ 复杂度的暴力解法中，很多问题可以通过双指针优化为 $O(n)$ 甚至更优的解法。

双指针主要分为以下几类：

- **对撞指针（Two Pointers towards each other)**：用于排序数组/字符串，双指针分别从两端向中间靠拢。常见问题：两数之和（Two Sum)、回文判断。
- **快慢指针（Fast and Slow Pointers)**：用于链表、数组，用于寻找循环、确定中间位置。常见问题：链表环检测、寻找链表中点。
- **滑动窗口（Sliding Window)**：用于处理子数组问题，如最长子串、不含重复字符的子数组。

2 双指针通用代码框架

2.1 对撞指针代码框架

适用于排序数组或字符串，通常用于查找满足某种条件的两个元素。

```
def two_pointers(nums, target):
    left, right = 0, len(nums) - 1  # 初始化两个指针
    while left < right:  # 指针相遇时终止
        current_sum = nums[left] + nums[right]
        if current_sum == target:
            return left, right  # 找到目标值
        elif current_sum < target:
            left += 1  # 左指针右移，使和增大
        else:
            right -= 1  # 右指针左移，使和减小
    return -1, -1  # 未找到
```

2.2 快慢指针代码框架

适用于链表问题，如判断链表是否有环，或找到中点。

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def hasCycle(head):
    slow, fast = head, head  # 初始化快慢指针
    while fast and fast.next:
        slow = slow.next  # 慢指针每次走一步
        fast = fast.next.next  # 快指针每次走两步
        if slow == fast:
            return True  # 相遇，说明有环
    return False
```

2.3 滑动窗口代码框架

适用于字符串或数组的子序列问题。

```
def sliding_window(s):
    left = 0  # 窗口左边界
    window = {}  # 记录窗口中的字符出现次数
    max_length = 0

    for right in range(len(s)):  # 窗口右边界扩展
        char = s[right]
        window[char] = window.get(char, 0) + 1

        while window[char] > 1:  # 发生重复，缩小窗口
            window[s[left]] -= 1
            left += 1

        max_length = max(max_length, right - left + 1)  # 记录最大长度

    return max_length
```

3 经典应用场景与例题

3.1 两数之和（对撞指针）

题目：在一个排序数组 nums 中找到两个数，使得它们的和等于目标值 target，返回它们的索引。
解法：使用双指针，一个从左，一个从右，逐步逼近目标值。

```
def twoSum(nums, target):
    left, right = 0, len(nums) - 1
    while left < right:
        if nums[left] + nums[right] == target:
            return [left, right]
        elif nums[left] + nums[right] < target:
            left += 1
        else:
            right -= 1
    return []
```

3.2 判断链表是否有环（快慢指针）

题目：判断链表是否包含环，即某个节点的 next 指针指向之前出现过的节点。

```
def hasCycle(head):
    slow, fast = head, head
    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next
        if slow == fast:
            return True
    return False
```

3.3 无重复字符的最长子串（滑动窗口）

题目：给定字符串 s，找出不含重复字符的最长子串的长度。

```
def lengthOfLongestSubstring(s):
    char_map = {}
    left = max_length = 0

    for right in range(len(s)):
        if s[right] in char_map:
            left = max(left, char_map[s[right]] + 1)
        char_map[s[right]] = right
        max_length = max(max_length, right - left + 1)

    return max_length
```

4 双指针问题的经典 LeetCode 题目

双指针（Two Pointers）是一种常见的算法技巧，适用于数组、字符串、链表等数据结构，主要用于优化时间复杂度，减少不必要的遍历。以下是 LeetCode 上的一些经典题目，按照 **对撞指针**、**快慢指针**、**滑动窗口**分类整理，并附上推荐的刷题顺序。

4.1 对撞指针（Two Pointers towards each other）

适用于 **有序数组**、**字符串**相关的问题，双指针分别从 **两端向中间靠拢**，逐步逼近答案。

题目编号	题目名称	难度
1	Two Sum（两数之和）(仅适用于已排序数组)	Easy
167	Two Sum II - Input Array Is Sorted（两数之和 II）	Easy
125	Valid Palindrome（验证回文串）	Easy
344	Reverse String（反转字符串）	Easy
15	3Sum（三数之和）	Medium
11	Container With Most Water（盛最多水的容器）	Medium
42	Trapping Rain Water（接雨水）	Hard

表 1: 经典对撞指针 LeetCode 题目

推荐做题顺序：

1. 基础对撞指针：125 → 344 → 167
2. 进阶对撞指针：1 → 15 → 11
3. 高难度对撞指针：42

4.2 快慢指针（Fast and Slow Pointers）

适用于 **链表**、**数组**，常用于 **寻找环**、**判断是否有环**、**寻找链表的中点等**。

题目编号	题目名称	难度
141	Linked List Cycle（环形链表）	Easy
876	Middle of the Linked List（链表的中间节点）	Easy
202	Happy Number（快乐数）	Easy
234	Palindrome Linked List（回文链表）	Medium
143	Reorder List（重排链表）	Medium
287	Find the Duplicate Number（寻找重复数）	Medium
142	Linked List Cycle II（环形链表 II - 找到环的起点）	Medium

表 2: 经典快慢指针 LeetCode 题目

推荐做题顺序：

1. 基础快慢指针：141 → 876 → 202
2. 进阶快慢指针：234 → 143 → 287
3. 高难度快慢指针：142

4.3 滑动窗口（Sliding Window）

适用于 **字符串**、**子数组问题**，在 **优化窗口大小**、**寻找最长/最短子串**时使用。

题目编号	题目名称	难度
3	Longest Substring Without Repeating Characters（无重复字符的最长子串）	Medium
438	Find All Anagrams in a String（找到字符串中所有字母异位词）	Medium
76	Minimum Window Substring（最小覆盖子串）	Hard
209	Minimum Size Subarray Sum（长度最小的子数组）	Medium
424	Longest Repeating Character Replacement（替换后的最长重复字符）	Medium
567	Permutation in String（字符串的排列）	Medium
30	Substring with Concatenation of All Words（所有单词的连接）	Hard

表 3: 经典滑动窗口 LeetCode 题目

推荐做题顺序：

1. 基础滑动窗口：3 → 209 → 424
2. 进阶滑动窗口：438 → 567 → 76
3. 高难度滑动窗口：30

5 总结

5.1 如何选择双指针解法？

- **数组或字符串中寻找两个数**：用 **对撞指针**（如 Two Sum II, 3Sum, Container With Most Water）。
- **链表是否有环、寻找链表中点**：用 **快慢指针**（如 Linked List Cycle, Middle of the Linked List）。
- **寻找最长/最短子串、优化子数组大小**：用 **滑动窗口**（如 Longest Substring Without Repeating Characters, Minimum Window Substring）。

5.2 推荐刷题顺序

1. **初学者**（对撞指针 + 快慢指针）：125 → 344 → 167 → 141 → 876
2. **进阶刷题**（更复杂的逻辑）：15 → 11 → 234 → 287 → 3 → 209
3. **挑战高难度**（优化时间复杂度）：42 → 142 → 76 → 30