

不能称之为算法的两根指针 Two Pointers

课程版本 v6.0

主讲 令狐冲



扫描二维码关注微信小程序/公众号
获取第一手求职资料

请在随课教程中预习如下内容：

什么是同向双指针？

什么是相向双指针？

双指针的鼻祖题 —— 两数之和 Two Sum

链表上的快慢指针算法

快速排序 & 归并排序

- 相向双指针
 - 几乎所有 Two Sum 变种
 - Partition
 - Quick Select
 - 分成两个部分
 - 分成三个部分
 - 一些你没听过的（但是面试会考的）排序算法
- 同向双指针

背向双指针

第一节课中的 Longest Palindromic Substring 的中心线枚举算法

第二节课中的 Find K Closest Elements

相向双指针

Two Sum 的一大类题（两位数的相关变形题）

Partition 的一大类题（两位数相关变形题）

同向双指针

滑动窗口类 Sliding Window

快慢指针类 Fast & Slow Pointers

等等

相向双指针

两根指针一头一尾，向中间靠拢直到相遇
时间复杂度 $O(n)$

相向双指针的问题主要可以分为下面三类：

1. Reverse 类（较少）
2. Two Sum 类（最多）
3. Partition 类（较多）

Reverse 类

上节课随课教程中的三步翻转法就是这一类

Valid Palindrome

验证一个字符串是否为回文串，忽略大小写和非英文字母字符

<http://www.lintcode.com/problem/valid-palindrome/>

<http://www.jiuzhang.com/solution/valid-palindrome/>

Follow up: 可以删掉一个字符

<http://www.lintcode.com/problem/valid-palindrome-ii/>

<https://www.jiuzhang.com/solution/valid-palindrome-ii/>

其实是一道证明题，证明见随课教程：

<http://www.jiuzhang.com/tutorial/algorithm/390>

Two Sum 类

先修内容中我们已经讲解了双指针的经典题 Two Sum
接下来我们来看这类问题可能的变化

只能使用 HashMap:

<http://www.lintcode.com/problem/two-sum-data-structure-design/>

<http://www.jiuzhang.com/solutions/two-sum-data-structure-design/>

使用 Two Pointers 会更快:

<http://www.lintcode.com/problem/two-sum-input-array-is-sorted/>

<http://www.jiuzhang.com/solutions/two-sum-input-array-is-sorted/>

Two Sum - Unique pairs

<https://www.lintcode.com/problem/two-sum-unique-pairs/>

<https://www.jiuzhang.com/solutions/two-sum-unique-pairs/>

问：是否可以先去重？

3Sum

<https://www.lintcode.com/problem/3sum/>

<https://www.jiuzhang.com/solutions/3sum/>

统计所有的和为 0 的三元组 (Triples)

Triangle Count

<https://www.lintcode.com/problem/triangle-count/>

<https://www.jiuzhang.com/solutions/triangle-count/>

统计所有和 $\leq \text{target}$ 的配对数

<http://www.lintcode.com/problem/two-sum-less-than-or-equal-to-target/>

<http://www.jiuzhang.com/solutions/two-sum-less-than-or-equal-to-target/>

统计所有和 $\geq \text{target}$ 的配对数

<http://www.lintcode.com/en/problem/two-sum-greater-than-target/>

<http://www.jiuzhang.com/solutions/two-sum-greater-than-target/>

Two Sum - Closest to Target

<https://www.lintcode.com/problem/two-sum-closest-to-target/>

<https://www.jiuzhang.com/solutions/two-sum-closest-to-target/>

Follow Up: 3Sum Closest

<http://www.lintcode.com/problem/3sum-closest/>

<http://www.jiuzhang.com/solutions/3sum-closest/>

- **4Sum**
- <http://www.lintcode.com/problem/4sum/>
- <http://www.jiuzhang.com/solutions/4sum/>

- **Two Sum - difference equals to target** (同向双指针, 讲解见随课教程)
- <http://www.lintcode.com/problem/two-sum-difference-equals-to-target/>
- <http://www.jiuzhang.com/solutions/two-sum-difference-equals-to-target/>

休息5分钟

take a break

Partition Array

<https://www.lintcode.com/problem/partition-array/>

<https://www.jiuzhang.com/solutions/partition-array/>

```
1 while left <= right:
2     while left <= right and nums[left] 应该在左侧:
3         left += 1
4     while left <= right and nums[right] 应该在右侧:
5         right -= 1
6
7     if left <= right:
8         # 找到了一个不该在左侧的和不该在右侧的, 交换他们
9         nums[left], nums[right] = nums[right], nums[left]
10        left += 1
11        right -= 1
```

独孤九剑 —— 破鞭式

时间复杂度与最内层循环主体的执行次数有关
与有多少重循环无关

Quick Select

随课教程: <http://www.jiuzhang.com/tutorial/algorithm/321>
<http://www.lintcode.com/problem/kth-smallest-numbers-in-unsorted-array/>
<http://www.lintcode.com/problem/kth-largest-element/>

- **Partition Array by Odd and Even**
 - <http://www.lintcode.com/problem/partition-array-by-odd-and-even/>
 - <http://www.jiuzhang.com/solutions/partition-array-by-odd-and-even/>
- **Interleaving Positive and Negative Numbers**
 - <http://www.lintcode.com/problem/interleaving-positive-and-negative-numbers/>
 - <http://www.jiuzhang.com/solutions/interleaving-positive-and-negative-integers/>
- **Sort Letters by Case**
 - <http://www.lintcode.com/problem/sort-letters-by-case/>
 - <http://www.jiuzhang.com/solutions/sort-letters-by-case/>

Sort Colors

<http://www.lintcode.com/problem/sort-colors/>

<http://www.jiuzhang.com/solutions/sort-colors/>

分成两个部分 vs 分成三个部分

随课教程 <http://www.jiuzhang.com/tutorial/algorithm/354>

☁️排序 Rainbow Sort

<https://www.lintcode.com/problem/sort-colors-ii/>

<https://www.jiuzhang.com/solutions/sort-colors-ii/>

问：猜一猜最优的时间复杂度？

烙饼排序 **Pancake Sort**（有可能会考哦）

https://en.wikipedia.org/wiki/Pancake_sorting

<http://www.geeksforgeeks.org/pancake-sorting/>

睡眠排序 **Sleep Sort**

https://rosettacode.org/wiki/Sorting_algorithms/Sleep_sort

面条排序 **Spaghetti Sort**

https://en.wikipedia.org/wiki/Spaghetti_sort

猴子排序 **Bogo Sort**

<https://en.wikipedia.org/wiki/Bogosort>

同向双指针

两根指针一前一后，直到前面的指针走过头
时间复杂度 $O(n)$

随课教程预习题

<http://www.lintcode.com/problem/window-sum/>

<http://www.lintcode.com/problem/remove-duplicate-numbers-in-array/>

<http://www.lintcode.com/problem/intersection-of-two-linked-lists/>

<http://www.lintcode.com/problem/linked-list-cycle/>

<http://www.lintcode.com/problem/linked-list-cycle-ii/>

<http://www.lintcode.com/problem/intersection-of-two-linked-lists/>

Two Sum Difference

见随课教程

<https://www.jiuzhang.com/tutorial/algorithm/359>

Move Zeroes

<http://www.lintcode.com/problem/move-zeroes/>

<http://www.jiuzhang.com/solution/move-zeroes>

将数组中非 0 的元素移动到数组的后半部分

Follow ups:

如果需要保证最少修改次数如何做？

不需要维持相对顺序 **vs** 需要维持相对顺序 算法有什么区别？

如果需要保证最少修改次数如何做？

使用 **Partition** 的方法可以做到交换次数最优

不需要维持相对顺序 **vs** 需要维持相对顺序 算法有什么区别？

不需要维护相对顺序，可以使用 **Partition** 的方法，时空复杂度和交换次数都是最优的

需要维护相对顺序的，只能使用同向双指针的方法，时空复杂度最优，但是交换次数不是最优的

更多同向双指针的难题

在《九章算法强化班》将进一步讲解
同向双指针的模板和其他双指针难题

随课教程: <http://www.jiuzhang.com/tutorial/algorithm/335>

三指针算法

烙饼排序

<http://www.lintcode.com/problem/pancake-sorting/>