王争的算法训练营

习题课：双指针&滑动窗口

技巧题

王争的算法训练营

配套习题： 双指针、滑动窗口、前后缀统计、位运算

344. 反转字符串

面试题 16.24. 数对和（例题1）

1. 两数之和

15. 三数之和

剑指 Offer 21. 调整数组顺序使奇数位于偶数前面

75. 颜色分类

283. 移动零 已排序未排序指针（例题2）

面试题 16.06. 最小差 类似合并两个有序数组（例题3）

面试题 17.11. 单词距离 类似合并两个有序数组

剑指 Offer 57 - II. 和为s的连续正数序列（例题1）

剑指 Offer 48. 最长不含重复字符的子字符串（例题2）

438. 找到字符串中所有字母异位词

76. 最小覆盖子串

53. 最大子序和（例题1）

121. 买卖股票的最佳时机（例题2）

238. 除自身以外数组的乘积（例题3）

面试题 05.03. 翻转数位

42. 接雨水

191. 位1的个数（例题1）

461. 汉明距离（例题2）

面试题 05.06. 整数转换

面试题 05.07. 配对交换

面试题 05.01. 插入

面试题 17.04. 消失的数字

剑指 Offer 56 - I. 数组中数字出现的次数

剑指 Offer 56 - II. 数组中数字出现的次数 II

面试题 16.01. 交换数字

231. 2 的幂

双指针

344. 反转字符串

面试题 16.24. 数对和（例题1）

1. 两数之和

15. 三数之和

剑指 Offer 21. 调整数组顺序使奇数位于偶数前面

75. 颜色分类

283. 移动零 已排序未排序指针（例题2）

面试题 16.06. 最小差 类似合并两个有序数组（例题3）

面试题 17.11. 单词距离 类似合并两个有序数组

剑指 Offer 57 - II. 和为s的连续正数序列 （例题1）

剑指 Offer 48. 最长不含重复字符的子字符串 （例题2）

438. 找到字符串中所有字母异位词

76. 最小覆盖子串

53. 最大子序和（例题1）

121. 买卖股票的最佳时机（例题2）

238. 除自身以外数组的乘积（例题3）

面试题 05.03. 翻转数位

42. 接雨水

191. 位1的个数 （例题1）

461. 汉明距离（例题2）

面试题 05.06. 整数转换

面试题 05.07. 配对交换

面试题 05.01. 插入

面试题 17.04. 消失的数字

剑指 Offer 56 - I. 数组中数字出现的次数

剑指 Offer 56 - II. 数组中数字出现的次数 II

面试题 16.01. 交换数字

231. 2 的幂

双指针

王争的算法训练营

344. 反转字符串

编写一个函数，其作用是将输入的字符串反转过来。输入字符串以字符数组 `char[]` 的形式给出。

不要给另外的数组分配额外的空间，你必须原地**修改输入数组**、使用 O(1) 的额外空间解决这一问题。

你可以假设数组中的所有字符都是 ASCII 码表中的可打印字符。

示例 1：

```
输入：["h","e","l","l","o"]
输出：["o","l","l","e","h"]
```

双指针

344. 反转字符串

```
class Solution {
    public void reverseString(char[] s) {
        int n = s.length;
        int i = 0;
        int j = n-1;
        while (i <= j) {
            char tmp = s[i];
            s[i] = s[j];
            s[j] = tmp;
            i++;
            j--;
        }
    }
}
```

面试题 16.24. 数对和（例题1）

设计一个算法，找出数组中两数之和为指定值的所有整数对。一个数只能属于一个数对。

示例 1:

```
输入: nums = [5,6,5], target = 11
输出: [[5,6]]
```

示例 2:

```
输入: nums = [5,6,5,6], target = 11
输出: [[5,6],[5,6]]
```

提示:

- `nums.length <= 100000`

双指针

王争的算法训练营

面试题 16.24. 数对和（例题1）

```
class Solution {
    public List<List<Integer>> pairSums(int[] nums, int target) {
        List<List<Integer>> results = new ArrayList<>();
        if (nums.length == 0) return results;
        Arrays.sort(nums);
        int i = 0;
        int j = nums.length-1;
        while (i < j) {
            if (nums[i]+nums[j] == target) {
                List<Integer> result = new ArrayList<>();
                result.add(nums[i]);
                result.add(nums[j]);
                results.add(result);
                i++;
                j--;
            } else if (nums[i]+nums[j] < target) {
                i++;
            } else {
                j--;
            }
        }
        return results;
    }
}
```

双指针

王争的算法训练营

1. 两数之和

给定一个整数数组 `nums` 和一个整数目标值 `target`，请你在该数组中找出 **和为目标值** *`target`* 的那 **两个** 整数，并返回它们的数组下标。

你可以假设每种输入只会对应一个答案。但是，数组中同一个元素在答案里不能重复出现。

你可以按任意顺序返回答案。

示例 1：

```
输入：nums = [2,7,11,15], target = 9
输出：[0,1]
解释：因为 nums[0] + nums[1] == 9 ，返回 [0, 1] 。
```

示例 2：

```
输入：nums = [3,2,4], target = 6
输出：[1,2]
```

示例 3：

```
输入：nums = [3,3], target = 6
输出：[0,1]
```

提示：

- $2 <= nums.length <= 10^4$
- $-10^9 <= nums[i] <= 10^9$
- $-10^9 <= target <= 10^9$
- **只会存在一个有效答案**

```java
class Solution {
    public int[] twoSum(int[] nums, int target) {
        int n = nums.length;
        int[] sortedNums = new int[n];
        for (int i = 0; i < n; ++i) {
            sortedNums[i] = nums[i];
        }
        Arrays.sort(sortedNums);
        boolean[] used = new boolean[n];
        int p = 0;
        int q = n-1;
        while (p < q) {
            int sum = sortedNums[p] + sortedNums[q];
            if (sum == target) {
                int oldp = find(nums, used, sortedNums[p]);
                int oldq = find(nums, used, sortedNums[q]);
                return new int[] {oldp, oldq};
            } else if (sum < target) {
                p++;
            } else {
                q--;
            }
        }
        return new int[0];
    }

    private int find(int[] nums, boolean[] used, int value) {
        int i = 0;
        while (i < nums.length) {
            if (nums[i] == value && used[i] == false) {
                used[i] = true;
                break;
            }
            i++;
        }
        return i;
    }
}
```

双指针

王争的算法训练营

15. 三数之和

给你一个包含 `n` 个整数的数组 `nums`，判断 `nums` 中是否存在三个元素 *a*，*b*，*c* ，使得 *a* + *b* + *c* = 0？请你找出所有和为 `0` 且不重复的三元组。

注意：答案中不可以包含重复的三元组。

示例 1：

```
输入：nums = [-1,0,1,2,-1,-4]
输出：[[-1,-1,2],[-1,0,1]]
```

示例 2：

```
输入：nums = []
输出：[]
```

示例 3：

```
输入：nums = [0]
输出：[]
```

提示：

- `0 <= nums.length <= 3000`
- `-10^5 <= nums[i] <= 10^5`

```java
class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
      Arrays.sort(nums);
      List<List<Integer>> result = new ArrayList<>();
      int n = nums.length;
      for (int i = 0; i < n; ++i) {
        if (i != 0 && nums[i] == nums[i-1]) continue; // 避免a重复
        int p = i+1;
        int q = n-1;
        while (p < q) {
          if (p>=i+2 && nums[p]==nums[p-1]) { // 避免b重复
            p++;
            continue;
          }
          if (q<=n-2 && nums[q]==nums[q+1]) { // 避免c重复
            q--;
            continue;
          }
          int sum = nums[p]+nums[q];
          if (sum == -1*nums[i]) {
            List<Integer> resultItem = new ArrayList<>();
            resultItem.add(nums[i]);
            resultItem.add(nums[p]);
            resultItem.add(nums[q]);
            result.add(resultItem);
            p++;
            q--;
          } else if (sum < -1*nums[i]) {
            p++;
          } else {
            q--;
          }
        }
      }
      return result;
    }
}
```

双指针

王争的算法训练营

剑指 Offer 21. 调整数组顺序使奇数位于偶数前面

输入一个整数数组，实现一个函数来调整该数组中数字的顺序，使得所有奇数位于数组的前半部分，所有偶数位于数组的后半部分。

示例：

```
输入：nums = [1,2,3,4]
输出：[1,3,2,4]
注：[3,1,2,4] 也是正确的答案之一。
```

提示：

1. `0 <= nums.length <= 50000`
2. `1 <= nums[i] <= 10000`

双指针

王争的算法训练营

剑指 Offer 21. 调整数组顺序使奇数位于偶数前面

```
class Solution {
    public int[] exchange(int[] nums) {
        int i = 0;
        int j = nums.length-1;
        while (i < j) {
            if (nums[i] % 2 == 1) {
                i++;
                continue;
            }
            if (nums[j] % 2 == 0) {
                j--;
                continue;
            }
            int tmp = nums[i];
            nums[i] = nums[j];
            nums[j] = tmp;
            i++;
            j--;
        }
        return nums;
    }
}
```

双指针

王争的算法训练营

75. 颜色分类

给定一个包含红色、白色和蓝色，一共 `n` 个元素的数组，原地对它们进行排序，使得相同颜色的元素相邻，并按照红色、白色、蓝色顺序排列。

此题中，我们使用整数 `0` 、 `1` 和 `2` 分别表示红色、白色和蓝色。

示例 1：

```
输入：nums = [2,0,2,1,1,0]
输出：[0,0,1,1,2,2]
```

示例 2：

```
输入：nums = [2,0,1]
输出：[0,1,2]
```

```java
class Solution {
    public void sortColors(int[] nums) {
        int p = 0;
        int q = nums.length-1;
        while (p < q) {
            if (nums[p] != 2) {
                p++;
                continue;
            }
            if (nums[q] == 2) {
                q--;
                continue;
            }
            swap(nums, p, q);
            p++;
            q--;
        }

        int i = 0;
        int j = p;
        if (nums[j] == 2) j--;
        while (i < j) {
            if (nums[i] == 0) {
                i++;
                continue;
            }
            if (nums[j] == 1) {
                j--;
                continue;
            }
            swap(nums, i, j);
            i++;
            j--;
        }
    }

    private void swap(int[] nums, int p, int q) {
        int tmp = nums[p];
        nums[p] = nums[q];
        nums[q] = tmp;
    }
}
```


283. 移动零 已排序未排序指针（例题2）

给定一个数组 `nums`，编写一个函数将所有 `0` 移动到数组的末尾，同时保持非零元素的相对顺序。

示例:

```
输入: [0,1,0,3,12]
输出: [1,3,12,0,0]
```

说明:

1. 必须在原数组上操作，不能拷贝额外的数组。
2. 尽量减少操作次数。

快速排序：partition()函数
选择排序：已排序未排序区间
插入排序：已排序未排序区

双指针

王争的算法训练营

283. 移动零 已排序未排序指针（例题2）

```
class Solution {
    public void moveZeroes(int[] nums) {
        int p = -1;
        int q = 0;
        while (q < nums.length) {
            if (nums[q] == 0) {
                q++;
                continue;
            }
            if (nums[q] != 0) {
                swap(nums, p+1, q);
                p++;
                q++;
            }
        }
    }

    private void swap(int[] nums, int i, int j) {
        int tmp = nums[i];
        nums[i] = nums[j];
        nums[j] = tmp;
    }
}
```

面试题 16.06. 最小差 类似合并两个有序数组（例题3）

给定两个整数数组 `a` 和 `b`，计算具有最小差绝对值的一对数值（每个数组中取一个值），并返回该对数值的差

1、暴力

2、双指针

示例：

```
输入：{1, 3, 15, 11, 2}, {23, 127, 235, 19, 8}
输出：3，即数值对(11, 8)
```

提示：

- `1 <= a.length, b.length <= 100000`
- `-2147483648 <= a[i], b[i] <= 2147483647`
- 正确结果在区间 `[0, 2147483647]` 内

面试题 16.06. 最小差 类似合并两个有序数组 **（例题3）**

```
class Solution {
    public int smallestDifference(int[] a, int[] b) {
        Arrays.sort(a);
        Arrays.sort(b);
        int n = a.length;
        int m = b.length;
        long minRet = Long.MAX_VALUE;
        int i = 0;
        int j = 0;
        while (i < n && j < m) {
            if (a[i] >= b[j]) {
                minRet = Math.min(minRet, (long)a[i]-b[j]);
                j++;
            } else {
                minRet = Math.min(minRet, (long)b[j]-a[i]);
                i++;
            }
        }
        return (int)minRet;
    }
}
```

面试题 17.11. 单词距离 类似合并两个有序数组

有个内含单词的超大文本文件，给定任意两个单词，找出在这个文件中这两个单词的最短距离(相隔单词数)。如果寻找过程在这个文件中会重复多次，而每次寻找的单词不同，你能对此优化吗?

示例：

```
输入：words =
["I","am","a","student","from","a","university","in","a","city"],
word1 = "a", word2 = "student"
输出：1
```

提示：

- `words.length <= 100000`

3 5 8 12

7 10 15

```java
class Solution {
    public int findClosest(String[] words, String word1, String word2) {
        List<Integer> w1ps = new ArrayList<>();
        List<Integer> w2ps = new ArrayList<>();
        for (int i = 0; i < words.length; ++i) {
            String word = words[i];
            if (word.equals(word1)) {
                w1ps.add(i);
            } else if (word.equals(word2)) {
                w2ps.add(i);
            }
        }
        int p1 = 0;
        int p2 = 0;
        int minRet = Integer.MAX_VALUE;
        while (p1 < w1ps.size() && p2 < w2ps.size()) {
            int pos1= w1ps.get(p1);
            int pos2 = w2ps.get(p2);
            if (pos1 > pos2) {
                if (minRet > pos1-pos2) {
                    minRet = pos1-pos2;
                }
                p2++;
            } else {
                if (minRet > pos2-pos1) {
                    minRet = pos2-pos1;
                }
                p1++;
            }
        }
        return minRet;
    }
}
```

滑动窗口

344. 反转字符串

面试题 16.24. 数对和（例题1）

1. 两数之和

15. 三数之和

剑指 Offer 21. 调整数组顺序使奇数位于偶数前面

75. 颜色分类

283. 移动零 已排序未排序指针（例题2）

面试题 16.06. 最小差 类似合并两个有序数组（例题3）

面试题 17.11. 单词距离 类似合并两个有序数组

剑指 Offer 57 - II. 和为s的连续正数序列（例题1）

剑指 Offer 48. 最长不含重复字符的子字符串（例题2）

438. 找到字符串中所有字母异位词

76. 最小覆盖子串

53. 最大子序和（例题1）

121. 买卖股票的最佳时机（例题2）

238. 除自身以外数组的乘积（例题3）

面试题 05.03. 翻转数位

42. 接雨水

191. 位1的个数（例题1）

461. 汉明距离（例题2）

面试题 05.06. 整数转换

面试题 05.07. 配对交换

面试题 05.01. 插入

面试题 17.04. 消失的数字

剑指 Offer 56 - I. 数组中数字出现的次数

剑指 Offer 56 - II. 数组中数字出现的次数 II

面试题 16.01. 交换数字

231. 2 的幂

滑动窗口

剑指 Offer 57 - II. 和为s的连续正数序列 （例题1）

输入一个正整数 `target` ，输出所有和为 `target` 的连续正整数序列（至少含有两个数）。

序列内的数字由小到大排列，不同序列按照首个数字从小到大排列。

示例 1：

```
输入：target = 9
输出：[[2,3,4],[4,5]]
```

示例 2：

```
输入：target = 15
输出：[[1,2,3,4,5],[4,5,6],[7,8]]
```

限制：

- `1 <= target <= 10^5`

```java
class Solution {
    public int[][] findContinuousSequence(int target) {
        List<int[]> result = new ArrayList<>();
        int p = 1;
        int q = 2;
        int sum = 3;
        while (p < q) {
            if (sum == target) {
                int[] arr = new int[q-p+1];
                for (int i = p; i <= q; ++i) {
                    arr[i-p] = i;
                }
                result.add(arr);
                sum -= p;
                p++;
                q++;
                sum += q;
            } else if (sum > target) {
                sum -= p;
                p++;
            } else {
                q++;
                sum += q;
            }
        }
        int[][] resultArr = new int[result.size()][];
        for (int i = 0; i < result.size(); ++i) {
            resultArr[i] = result.get(i);
        }
        return resultArr;
    }
}
```

滑动窗口

剑指 Offer 48. 最长不含重复字符的子字符串 （例题2）

请从字符串中找出一个最长的不包含重复字符的子字符串，计算该最长子字符串的长度。

示例 1:

```
输入: "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: "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: "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：

- `s.length <= 40000`

滑动窗口

王争的算法训练营

剑指 Offer 48. 最长不含重复字符的子字符串 （例题2）

```
class Solution {
    public int lengthOfLongestSubstring(String s) {
        int n = s.length();
        if (n == 0) return 0;
        int p = 0;
        int q = 0;
        Set<Character> set = new HashSet<>();
        int maxLen = 0;
        while (q < n) {
            char c = s.charAt(q);
            if (!set.contains(c)) {
                set.add(c);
                q++;
                if (q-p > maxLen) maxLen = q-p;
                continue;
            }
            while (set.contains(c)) {
                set.remove(s.charAt(p));
                p++;
            }
        }
        return maxLen;
    }
}
```

滑动窗口

王争的算法训练营

438. 找到字符串中所有字母异位词

给定一个字符串 **s** 和一个非空字符串 **p**，找到 **s** 中所有是 **p** 的字母异位词的子串，返回这些子串的起始索引。

字符串只包含小写英文字母，并且字符串 **s** 和 **p** 的长度都不超过 20100。

说明：

- 字母异位词指字母相同，但排列不同的字符串。
- 不考虑答案输出的顺序。

示例 1:

```
输入:
s: "cbaebabacd" p: "abc"

输出:
[0, 6]

解释:
起始索引等于 0 的子串是 "cba", 它是 "abc" 的字母异位词。
起始索引等于 6 的子串是 "bac", 它是 "abc" 的字母异位词。
```

示例 2:

```
输入:
s: "abab" p: "ab"

输出:
[0, 1, 2]

解释:
起始索引等于 0 的子串是 "ab", 它是 "ab" 的字母异位词。
起始索引等于 1 的子串是 "ba", 它是 "ab" 的字母异位词。
起始索引等于 2 的子串是 "ab", 它是 "ab" 的字母异位词。
```

1、找出所有的子串

2、怎么判断子串跟p是异位词

```java
class Solution {
    public List<Integer> findAnagrams(String s, String p) {
        int n = s.length();
        int m = p.length();
        if (m > n) return new ArrayList<>();
        int[] needs = new int[26];
        for (int i = 0; i < m; ++i) {
            needs[p.charAt(i)-'a']++;
        }
        int[] matched = new int[26];

        int startp = 0;
        int endp = 0;
        List<Integer> result = new ArrayList<>();
        while (endp < m) {
            matched[s.charAt(endp)-'a']++;
            endp++;
        }
        if (same(needs, matched)) {
            result.add(startp);
        }

        while (endp < n && startp < n) {
            matched[s.charAt(startp)-'a']--;
            matched[s.charAt(endp)-'a']++;
            startp++;
            endp++;
            if (same(needs, matched)) {
                result.add(startp);
            }
        }

        return result;
    }
}
```

```java
private boolean same(int[] needs, int[] matched) {
    for (int i = 0; i < needs.length; ++i) {
        if (needs[i] != matched[i]) return false;
    }
    return true;
}
```

滑动窗口

76. 最小覆盖子串 （困难）

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： 如果 `s` 中存在这样的子串，我们保证它是唯一的答案。

滑动窗口[l, r]

1、如果没有覆盖，r++

2、如果有覆盖，对比是否最小，l++

循环上面两个步骤，直到r>=n

怎么判断滑动窗口覆盖了t?

示例 1：

```
输入：s = "ADOBECODEBANC", t = "ABC"
输出："BANC"
```

示例 2：

```
输入：s = "a", t = "a"
输出："a"
```

提示：

- `1 <= s.length, t.length <= 10^5`
- `s` 和 `t` 由英文字母组成

```java
class Solution {
    public String minWindow(String s, String t) {
        int minWSize = Integer.MAX_VALUE;
        int minWStart = -1;
        int minWEnd = -1;
        Map<Character, Integer> tmap = new HashMap<>(); //模式串
        Map<Character, Integer> wmap = new HashMap<>(); //滑动窗口
        for (int i = 0; i < t.length(); ++i) {
            int count = 1;
            if (tmap.containsKey(t.charAt(i))) {
                count += tmap.get(t.charAt(i));
            }
            tmap.put(t.charAt(i), count);
        }
        int n = s.length();
        int l = 0;
        int r = -1;
        while (l<n && r<n) {
            while (!match(wmap, tmap)) {
                r++;
                if (r > n-1) {
                    break;
                }
                char c = s.charAt(r);
                if (tmap.containsKey(c)) {
                    int count = 1;
                    if (wmap.containsKey(c)) {
                        count += wmap.get(c);
                    }
                    wmap.put(c, count);
                }
            }
            if (match(wmap, tmap)) {
                if (minWSize > r-l+1) {
                    minWSize = r-l+1;
                    minWStart = l;
                    minWEnd = r;
                }
                char c = s.charAt(l);
                if (tmap.containsKey(c)) {
                    int count = wmap.get(c);
                    if (count-1 == 0) {
                        wmap.remove(c);
                    } else {
                        wmap.put(c, count-1);
                    }
                }
                l++;
            }
        }
        if (minWStart == -1) return "";
        return s.substring(minWStart, minWEnd+1);
    }
}
```

```java
private boolean match(Map<Character, Integer> wmap, Map<Character, Integer> tmap) {
    for (Map.Entry<Character, Integer> entry : tmap.entrySet()) {
        Character key = entry.getKey();
        if (!wmap.containsKey(key)) return false;
        if (wmap.get(key) < entry.getValue()) return false;
    }
    return true;
}
```

提问环节

关注微信公众号“**小争哥**”，

后台回复“**PDF**”获取独家算法资料