

## 1. CodeExercise:

1. 简介:

2. 日常刷题:

1. 2022.6.21——数组

1. 26. 删除有序数组中的重复项
2. 83. 删除排序链表中的重复元素
3. 27. 移除元素
4. 283. 移动零
5. 167. 两数之和 II - 输入有序数组
6. 344. 反转字符串
7. 5. 最长回文子串

3. 补充内容:

# CodeExercise:

---

## 简介:

---

记录从大一暑假开始的刷题之旅。

**PS:** 这个代码库保存了我从大一入学所刷的所有算法题，只是从**2022年6月21日**开始整理。

---

## 日常刷题:

---

## 2022.6.21——数组

周二——雷阵雨转阴——35℃/22℃

今天开始刷LeetCode中的《初涉算法》篇，先从数组开始:

### 26. 删除有序数组中的重复项

类型: 数组 双指针

```

class Solution {
public:
    int removeDuplicates(vector<int>& nums) {
        int fast = 0, slow = 0;
        while(fast < nums.size()) {
            if(nums[fast++] != nums[slow]) {
                nums[++slow] = nums[fast-1];
            }
        }
        return slow+1;
    }
};

```

时间复杂度：  $O(N)$  空间复杂度：  $O(1)$

使用了**fast**和**slow**两个快慢指针进行判断，每次循环快指针都会向前走一位，通过快指针指向的位置和慢指针比较，若和慢指针所指向的数不同则慢指针走一位并将快指针所对应的值赋给慢指针指向的位置。

最后慢指针所指向的位置则为数组长度-1，所以返回**slow+1**即可。

详情参考：[双指针技巧秒杀七道数组题目 :: labuladong](#)的算法小抄

由上一题可延伸：

### 83. 删除排序链表中的重复元素

类型： 链表

```

class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        if(head==nullptr) return nullptr;
        ListNode *slow = head, *fast = head;
        while(fast->next!=nullptr) {
            fast = fast->next;
            if(fast->val!=slow->val) {
                slow = slow->next;
                slow->val = fast->val;
            }
        }
        slow->next = nullptr;
        return head;
    }
};

```

时间复杂度：  $O(N)$  空间复杂度：  $O(1)$

同样使用快慢指针，详情见上文。

## 27. 移除元素

类型： 数组 双指针

```
class Solution {
public:
    int removeElement(vector<int>& nums, int val) {
        int slow = 0, fast = 0;
        while(fast != nums.size()) {
            if(nums[fast++] != val) {
                nums[slow++] = nums[fast-1];
            }
        }
        return slow;
    }
};
```

时间复杂度：  $O(N)$  空间复杂度：  $O(1)$

同样利用双指针，将重复的元素覆盖，最后返回数组长度为slow的值。

## 283. 移动零

类型： 数组 双指针

```
class Solution {
public:
    void moveZeroes(vector<int>& nums) {
        int slow = 0, fast = 0;
        while(fast != nums.size()) {
            if(nums[fast++] != 0) {
                nums[slow++] = nums[fast-1];
            }
        }
        while(slow != nums.size()) {
            nums[slow++] = 0;
        }
    }
};
```

时间复杂度：  $O(N)$  空间复杂度：  $O(1)$

利用上一题的做法最后在将slow指针往后的多余数替换为0。

## 167. 两数之和 II - 输入有序数组

类型： 数组 双指针 二分查找

```
class Solution {
public:
    vector<int> twoSum(vector<int>& numbers, int target) {
        int left = 0, right = numbers.size()-1;
        while(left < right) {
            int sum = numbers[left] + numbers[right];
            if(sum == target) {
                return {left+1, right+1};
            } else if(sum < target) {
                left++;
            } else {
                right--;
            }
        }
        return {0, 0};
    }
};
```

时间复杂度：  $O(n \log n)$  空间复杂度：  $O(1)$

通过二分查找的方法先从数组的两端的和开始比较，若`sum`比`target`小，则`left+1`；若`sum`比`target`大，则`right-1`。

### 344. 反转字符串

类型： 递归 双指针 字符串

```
class Solution {
public:
    void reverseString(vector<char>& s) {
        int left = 0, right = s.size() - 1;
        while(left < right) {
            char temp = s[left];
            s[left] = s[right];
            s[right] = temp;
            left++; right--;
        }
    }
};
```

时间复杂度：  $O(N)$  空间复杂度：  $O(1)$

通过双指针从头和尾同时进行。

## 5. 最长回文子串

类型： 字符串 动态规划

```
class Solution {
public:
    string longestPalindrome(string s) {
        string ans = "";
        for(int i=0;i<s.length();i++) {
            string s1 = charge(s,i,i);
            string s2 = charge(s,i,i+1);
            ans = ans.length() > s1.length() ? ans : s1;
            ans = ans.length() > s2.length() ? ans : s2;
        }
        return ans;
    }
    string charge(string s,int left,int right) {
        while(left>=0 && right < s.length() && s[left]==s[right]) {
            left--;
            right++;
        }
        return s.substr(left+1,right-left-1);
    }
};
```

时间复杂度：  $O(n^2)$  空间复杂度：  $O(1)$

先确定中心点的位置，计算回文字符串长度分别为奇数和偶数的情况，利用双指针求出每一个回文字符串，并通过比较实时更新最长回文字符串。

---

## 补充内容：

---