王争的算法训练营

习题课：纯编程题

配套习题(11)：

1. 两数之和（简单） 两层for循环

1108. IP 地址无效化（简单） 简单字符串替换

344. 反转字符串（简单）真的就是反转字符串

剑指 Offer 58 - I. 翻转单词顺序（简单） 反转两次

125. 验证回文串 （简单） 比普通的验证回文串稍微复杂一点点

9. 回文数（简单）需要先将数字转化成字符串数组

58. 最后一个单词的长度（简单） 从后往前扫描更简单

剑指 Offer 05. 替换空格（简单） 字符串中元素替换，减少数组元素的搬移

剑指 Offer 58 - II. 左旋转字符串（简单） 纯数组搬移数据 题解有技巧的不用看 入门练练手

26. 删除排序数组中的重复项（简单）顺序扫描 下标操作

剑指 Offer 67. 把字符串转换成整数（中等）经典atoi()，注意范围越界处理

题型说明：

很常考，在算法面试中比重很大，是重中之重

解决思路简单，但编程实现繁琐

纯编程题，也叫模拟题，或者翻译题，纯粹考察编程实现能力，基本不涉及任何数据结构和算法，比如实现一个atoi()字符串转整数的函数。

一看就知道怎么做，但实现起来比较繁琐。边界条件处理复杂，需要考虑的特殊情况比较多，容易写出Bug。

如何准备这类题型?

没有固定套路和细分题型，攻克它的关键：多练->提高编程能力

解题技巧：

a）先忽略掉不容易处理的特殊情况，只考虑正常情况，简化编程。

b）写代码前先写注释，通过注释让代码模块化，让思路更清晰。

c）写完代码多举几个特例，来验证代码是否正确。

习题课：纯编程题

1. 两数之和（简单） 两层for循环

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** 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] 。
```

1. 两数之和（简单） 两层for循环

时间复杂度：O(n^2)，空间复杂度：O(1)

```
class Solution {
    public int[] twoSum(int[] nums, int target) {
      // 查找组合(A, B)满足A+B=target
      for (int i = 0 ; i < nums.length; ++i) { // 先确定A
        // B的下标总是比A的下标大，避免类似(1,5)(5,1)这样的重复组合
        for (int j = i+1; j < nums.length; ++j) { // 再确定B
          if (nums[i]+nums[j] == target) {
            return new int[] {i, j}; // 只有一个答案，找到就可以返回了
          }
        }
      }
      return null;
    }
}
```

纯编程题

1108. IP 地址无效化（简单）

1108. IP 地址无效化

难度 简单　65　收藏　分享　切换为英文　接收动态　反馈

给你一个有效的 IPv4 地址 `address`，返回这个 IP 地址的无效化版本。

所谓无效化 IP 地址，其实就是用 `"[.]"` 代替了每个 `"."`。

示例 1：

```
输入：address = "1.1.1.1"
输出："1[.]1[.]1[.]1"
```

示例 2：

```
输入：address = "255.100.50.0"
输出："255[.]100[.]50[.]0"
```

提示：

- 给出的 `address` 是一个有效的 IPv4 地址

纯编程题

1108. IP 地址无效化（简单）

```
class Solution {
    public String defangIPaddr(String address) {
        return address.replace(".", "[.]");
    }
}
```

1108. IP 地址无效化（简单）

时间复杂度：O(n)，空间复杂度：O(n)

```
class Solution {
    public String defangIPaddr(String address) {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < address.length(); ++i) {
            char c = address.charAt(i);
            if (c != '.') {
                sb.append(c);
            } else {
                sb.append("[.]");
            }
        }
        return sb.toString();
    }
}
```

支持可扩容的字符类型的数组

1108. IP 地址无效化（简单）

时间复杂度：O(n)，空间复杂度：O(n)

```
class Solution {
    public String defangIPaddr(String address) {
        char[] origin = address.toCharArray();
        int n = origin.length;
        int newN = n+2*3;
        char[] newString  = new char[newN];
        int k = 0;
        for (int i = 0; i < n; i++) {
            if (origin[i] != '.') {
                newString[k] = origin[i];
                k++;
            } else {
                newString[k++] = '[';
                newString[k++] = '.';
                newString[k++] = ']';
            }
        }
        return new String(newString);
    }
}
```

固定长度的数组

习题课：纯编程题

344. 反转字符串（简单）真的就是反转字符串

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"]
```

示例 2：

```
输入：["H","a","n","n","a","h"]
输出：["h","a","n","n","a","H"]
```

```
class Solution {
    public void reverseString(char[] s) {
        int n = s.length;
        // 这里下标到底是n/2还是n/2+1可以举几个例子（奇偶）验证一下即可
        for (int i = 0; i < n/2; i++) {
            // 交换数组中两个元素的标准写法
            char tmp = s[i];
            s[i] = s[n-i-1];
            s[n-i-1] = tmp;
        }
    }
}
```

时间复杂度：O(n)，空间复杂度：O(1)

```
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--;
        }
    }
}
```

时间复杂度：O(n)，空间复杂度：O(1)

习题课：纯编程题

王争的算法训练营

剑指 Offer 58 - I. 翻转单词顺序（简单）

输入一个英文句子，翻转句子中单词的顺序，但单词内字符的顺序不变。为简单起见，标点符号和普通字母一样处理。例如输入字符串"I am a student. "，则输出"student. a am I"。

1. 先split->空间复杂度高O(n)
2. 两次反转->空间复杂度O(1)，原地反转

示例 1：

```
输入: "the sky is blue"
输出: "blue is sky the"
```

示例 2：

```
输入: "  hello world!  "
输出: "world! hello"
解释: 输入字符串可以在前面或者后面包含多余的空格，但是反转后的字符不能包括。
```

示例 3：

```
输入: "a good   example"
输出: "example good a"
解释: 如果两个单词间有多余的空格，将反转后单词间的空格减少到只含一个。
```

说明：

- 无空格字符构成一个单词。
- 输入字符串可以在前面或者后面包含多余的空格，但是反转后的字符不能包括。
- 如果两个单词间有多余的空格，将反转后单词间的空格减少到只含一个。

时间复杂度：O(n)，空间复杂度：O(1)

```java
class Solution {
    public String reverseWords(String s) {
        // 这里只是因为Java中的String不可变
        char[] str = s.toCharArray();

        int n = trim(str); //自己实现的
        if (n == 0) return "";
        reverse(str, 0, n-1);
        int p = 0;
        while (p < n) {
            int r = p;
            while (r < n && str[r] != ' ') {
                r++;
            }
            reverse(str, p, r-1);
            p = r+1;
        }

        // 这里只是为了配合输出
        char[] newStr = new char[n];
        for (int i = 0; i < n; ++i) {
            newStr[i] = str[i];
        }
        return new String(newStr);
    }
}
```

```java
// 原地删除前置空格和后置空格，以及内部多于的空格，返回新字符串长度
private int trim(char[] str) {
    int n = str.length;
    // 跳过首空格
    int i = 0;
    while (i < n && str[i] == ' ') {
        i++;
    }
    // 跳过尾空格
    int j = n-1;
    while (j >= 0 && str[j] == ' ') {
        j--;
    }
    // 清除内部多于空格，并且把i~j之间的字符搬移到数组最前面
    int k = 0;
    while (i <= j) {
        if (str[i] == ' ') {
            if (i+1<=j && str[i+1] != ' ') {
                str[k++] = ' ';
            }
        } else {
            str[k++] = str[i];
        }
        i++;
    }
    return k;
  }
}
```

习题课：纯编程题

125. 验证回文串 （简单） 比普通的验证回文串稍微复杂一点点

给定一个字符串，验证它是否是回文串，只考虑字母和数字字符，可以忽略字母的大小写。

说明：本题中，我们将空字符串定义为有效的回文串。

1. 先格式化：扫描两遍
2. 不用先格式化：扫描一遍

示例 1:

```
输入: "A man, a plan, a canal: Panama"
输出: true
```

示例 2:

```
输入: "race a car"
输出: false
```

```java
class Solution {
    public boolean isPalindrome(String s) {
        int i = 0;
        int j = s.length()-1;
        // 双指针i,j
        while (i < j) {
            // 不是数字或字母的话，i就一直++
            if (!isAlpha(s.charAt(i))) {
                i++;
                continue;
            }
            // 不是数字或字母的话，j就一直--
            if (!isAlpha(s.charAt(j))) {
                j--;
                continue;
            }
            // 走到这里的话，i、j都指向数字或字母，看看两个字符是否相同
            if (toLower(s.charAt(i)) != toLower(s.charAt(j))) {
                return false;
            } else {
              // i和j往中间挪一位
              i++;
              j--;
            }
        }
        return true;
    }

    // 大写转小写
    private char toLower(char c) {
        if (c >= 'a' && c <= 'z') return c;
        if (c >= '0' && c <= '9') return c;
        //大写A~Z 65~90，小写a~z 97~122
        return (char)((int)c+32);
    }

    // 判断是不是数字或字母
    private boolean isAlpha(char c) {
        if (c >= 'a' && c <= 'z') return true;
        if (c >= 'A' && c <= 'Z') return true;
        if (c >= '0' && c <= '9') return true;
        return false;
    }
}
```

时间复杂度：O(n)，空间复杂度：O(1)

习题课：纯编程题

王争的算法训练营

9. 回文数（简单）需要先将数字转化成字符串数组

1.数字转字符串
2.字符串转数字

给你一个整数 `x` ，如果 `x` 是一个回文整数，返回 `true` ；否则，返回 `false` 。

回文数是指正序（从左向右）和倒序（从右向左）读都是一样的整数。例如， `121` 是回文，而 `123` 不是。

示例 1：

```
输入：x = 121
输出：true
```

示例 2：

```
输入：x = -121
输出：false
解释：从左向右读，为 -121 。 从右向左读，为 121- 。因此它不是一个回文数。
```

示例 3：

```
输入：x = 10
输出：false
解释：从右向左读，为 01 。因此它不是一个回文数。
```

示例 4：

```
输入：x = -101
输出：false
```

提示：

- $-2^{31} <= x <= 2^{31} - 1$

习题课：纯编程题

王争的算法训练营

```
class Solution {
    public boolean isPalindrome(int x) {
        // -2147483648 ~2147483647
        int[] digits = new int[10];
        if (x < 0) return false;
        int k = 0;
        // 将x转化成字符串数组
        while (x != 0) {
            digits[k] = x % 10;
            x = x / 10;
            k++;
        }
        // 判断回文串
        for (int i = 0; i < k/2; ++i) { // 举例验证
            if (digits[i] != digits[k-i-1]) { // 举例验证
                return false;
            }
        }
        return true;
    }
}
```

时间复杂度：O(1)，空间复杂度：O(1)

```
class Solution {
    public boolean isPalindrome(int x) {
        // -2147483648 ~2147483647
        if (x < 0) return false;
        int backupX = x;
        int y = 0;//y为x反转之后的值
        while (x != 0) {//将x转化成字符串数组的过程计算y
            y = y*10 + x % 10;
            x = x / 10;
        }
        return backupX == y;
    }
}
```

时间复杂度：O(n)，空间复杂度：O(1)

习题课：纯编程题

58. 最后一个单词的长度（简单） 从后往前扫描更简单

给你一个字符串 `s`，由若干单词组成，单词之间用空格隔开。返回字符串中最后一个单词的长度。如果不存在最后一个单词，请返回 0 。

单词 是指仅由字母组成、不包含任何空格字符的最大子字符串。

示例 1：

```
输入：s = "Hello World"
输出：5
```

示例 2：

```
输入：s = "   "
输出：0
```

提示：

- `1 <= s.length <= 10^4`
- `s` 仅有英文字母和空格 `' '` 组成

习题课：纯编程题

王争的算法训练营

时间复杂度：O(n)，空间复杂度：O(1)

```
class Solution {
    // 从后往前扫描更好
    public int lengthOfLastWord(String s) {
        // 去掉后置空格
        int n = s.length();
        int i = n-1;
        while (i >= 0 && s.charAt(i)==' ') {
            i--;
        }
        if (i < 0) return 0;
        //遍历找到分割点空格，统计最后一个单词长度
        int len = 0;
        while (i >= 0 && s.charAt(i)!= ' ') {
            len++;
            i--;
        }
        return len;
    }
}
```

习题课：纯编程题

剑指 Offer 05. 替换空格（简单） 字符串中元素替换，减少数组元素的搬移

请实现一个函数，把字符串 `s` 中的每个空格替换成"%20"。

跟“IP地址无效化”相同

示例 1：

```
输入：s = "We are happy."
输出："We%20are%20happy."
```

限制：

`0 <= s 的长度 <= 10000`

```java
class Solution {
    // 不使用Java提供的ArrayList和StringBuilder
    public String replaceSpace(String s) {
        int n = s.length();
        // 统计有多个空格
        int spaceCount = 0;
        for (int i = 0; i < n; ++i) {
            if (s.charAt(i)==' ') {
                spaceCount++;
            }
        }
        // 计算新数组长度
        int newLen = n + spaceCount * 2;
        char[] newStr = new char[newLen];
        int j = newLen-1;
        for (int i = n-1; i >= 0; --i) {
            char c = s.charAt(i);
            if (c != ' ') {
                newStr[j] = c;
                j--;
            } else {
                newStr[j] = '0';
                newStr[j-1] = '2';
                newStr[j-2] = '%';
                j-=3;
            }
        }
        return new String(newStr);
    }
}
```

时间复杂度：O(n)，空间复杂度：O(n)

习题课：纯编程题

王争的算法训练营

[剑指 Offer 58 - II. 左旋转字符串](#)（简单） 纯数组搬移数据 题解有技巧的不用看 入门练练手

字符串的左旋转操作是把字符串前面的若干个字符转移到字符串的尾部。请定义一个函数实现字符串左旋转操作的功能。比如，输入字符串"abcdefg"和数字2，该函数将返回左旋转两位得到的结果"cdefgab"。

3种解法

示例 1：

```
输入: s = "abcdefg", k = 2
输出: "cdefgab"
```

示例 2：

```
输入: s = "lrloseumgh", k = 6
输出: "umghlrlose"
```

限制：

- `1 <= k < s.length <= 10000`

习题课：纯编程题

剑指 Offer 58 - II. 左旋转字符串（简单） 纯数组搬移数据 题解有技巧的不用看 入门练练手

```
class Solution {
    // 往左移动n位
    public String reverseLeftWords(String s, int n) {
        char[] tmp = new char[s.length()];
        // 数组分为[0~n~len),先把0~n-1放到tmp后面
        for (int i = 0; i < n; ++i) {
            tmp[i+(s.length()-n)] = s.charAt(i);
        }
        // 再把n~len-1放到tmp的前面
        for (int i = n; i < s.length(); i++) {
            tmp[i-n] = s.charAt(i);
        }
        return new String(tmp);
    }
```

时间复杂度：O(len)，空间复杂度：O(len)，len表示字符串长度

习题课：纯编程题

剑指 Offer 58 - II. 左旋转字符串（简单） 纯数组搬移数据 题解有技巧的不用看 入门练练手

```
class Solution {
    // 往左移动n位
    public String reverseLeftWords(String s, int n) {
        char[] str = s.toCharArray();
        char[] tmp = new char[n];
        for (int i = 0; i < n; ++i) {
            tmp[i] = str[i];
        }
        for (int i = n; i < s.length(); ++i) {
            str[i-n] = str[i];
        }
        for (int i = 0; i < n; ++i) {
            str[s.length()-n+i] = tmp[i];
        }
        return new String(str);
    }
}
```

时间复杂度：O(len)，空间复杂度：O(n)，len表示字符串长度

习题课：纯编程题

剑指 Offer 58 - II. 左旋转字符串（简单） 纯数组搬移数据 题解有技巧的不用看 入门练练手

```
class Solution {
    // 往左移动n位
    public String reverseLeftWords(String s, int n) {
        char[] str = s.toCharArray();
        for (int i = 0; i < n; ++i) { // 移动n次,每次左移1位
            char tmp = str[0];
            for (int j = 1; j < str.length; ++j) {
                str[j-1] = str[j];
            }
            str[str.length-1] = tmp;
        }
        return new String(str);
    }
}
```

时间复杂度：O(len*n)，空间复杂度：O(1)，len表示字符串长度

习题课：纯编程题

王争的算法训练营

26. 删除排序数组中的重复项（简单）顺序扫描 下标操作

给你一个有序数组 `nums` ，请你 原地 删除重复出现的元素，使每个元素 **只出现一次** ，返回删除后数组的新长度。

不要使用额外的数组空间，你必须在 原地 **修改输入数组** 并在使用 O(1) 额外空间的条件下完成。

两种解法：
新开数组、原地

说明：

为什么返回数值是整数，但输出的答案是数组呢？

请注意，输入数组是以**「引用」**方式传递的，这意味着在函数里修改输入数组对于调用者是可见的。

你可以想象内部操作如下：

```
// nums 是以“引用”方式传递的。也就是说，不对实参做任何拷贝
int len = removeDuplicates(nums);

// 在函数里修改输入数组对于调用者是可见的。
// 根据你的函数返回的长度, 它会打印出数组中 该长度范围内 的所有元素。
for (int i = 0; i < len; i++) {
    print(nums[i]);
}
```

示例 1：

```
输入：nums = [1,1,2]
输出：2, nums = [1,2]
解释：函数应该返回新的长度 2 ，并且原数组 nums 的前两个元素被修改为 1, 2 。不需要考虑数组中超出新长度后面的元素。
```

```java
class Solution {
    public int removeDuplicates(int[] nums) {
        int n = nums.length;
        int k = 0;//[0,k]栈中元素
        for (int i = 1; i < n; ++i) {
            if (nums[i] != nums[k]) {//放入栈
                k++;
                nums[k] = nums[i];
            }
        }
        return k+1;
    }
}
```

时间复杂度：O(n)，空间复杂度：O(1)

剑指 Offer 67. 把字符串转换成整数（中等）经典atoi()，注意范围越界处理

首先，该函数会根据需要丢弃无用的开头空格字符，直到寻找到第一个非空格的字符为止。

当我们寻找到的第一个非空字符为正或者负号时，则将该符号与之后面尽可能多的连续数字组合起来，作为该整数的正负号；假如第一个非空字符是数字，则直接将其与之后连续的数字字符组合起来，形成整数。

该字符串除了有效的整数部分之后也可能会存在多余的字符，这些字符可以被忽略，它们对于函数不应该造成影响。

注意：假如该字符串中的第一个非空格字符不是一个有效整数字符、字符串为空或字符串仅包含空白字符时，则你的函数不需要进行转换。

在任何情况下，若函数不能进行有效的转换时，请返回 0。

说明：

假设我们的环境只能存储 32 位大小的有符号整数，那么其数值范围为 $[-2^{31}, 2^{31} - 1]$。如果数值超过这个范围，请返回 INT_MAX $(2^{31} - 1)$ 或 INT_MIN (-2^{31}) 。

习题课：纯编程题

王争的算法训练营

剑指 Offer 67. 把字符串转换成整数（中等）经典atoi()，注意范围越界处理

示例 1:

```
输入: "42"
输出: 42
```

示例 2:

```
输入: "   -42"
输出: -42
解释: 第一个非空白字符为 '-', 它是一个负号。
     我们尽可能将负号与后面所有连续出现的数字组合起来，最后得到 -42 。
```

示例 3:

```
输入: "4193 with words"
输出: 4193
解释: 转换截止于数字 '3' ，因为它的下一个字符不为数字。
```

示例 4:

```
输入: "words and 987"
输出: 0
解释: 第一个非空字符是 'w', 但它不是数字或正、负号。
     因此无法执行有效的转换。
```

示例 5:

```
输入: "-91283472332"
输出: -2147483648
解释: 数字 "-91283472332" 超过 32 位有符号整数范围。
     因此返回 INT_MIN (−2^31) 。
```

时间复杂度：O(n)，空间复杂度：O(1)

```
class Solution {
    public int strToInt(String str) {
        char[] chars = str.toCharArray();
        int n = chars.length;

        // 处理空
        if (n == 0) return 0;

        // 处理前置空格
        int i = 0;
        while (i < n && chars[i] == ' ') {
            i++;
        }

        // 全为空格
        if (i == n) return 0;

        // 处理符号
        int sign = 1;
        char c = chars[i];
        if (c == '-') {
            sign = -1;
            i++;
        } else if (c == '+') {
            sign = 1;
            i++;
        }
        //…下一页
    }
}
```

```
                                    // "  -42" 前导空格，符号位
                                    // "4193abc" 后置非数字字符
                                    // "abc987" 返回0
                                    // "-9182838383838" 超过整数范围 返回Integer.MIN_VALUE
                                    // “9182838383838” 超过整数范围 返回Integer.MAX_VALUE
                                    // "" 空字符串 返回0
                                    // "  " 全文空格 返回0
```

```
// 真正处理数字
// 整数范围-2147483648~2147483647
int intAbsHigh = 214748364;
int result = 0;
while (i < n && chars[i] >= '0' && chars[i] <= '9') {
    int d = chars[i]-'0';
    // 判断再乘以10，加d之后，是否越界
    if (result > intAbsHigh) {
        if (sign==1) return Integer.MAX_VALUE; 214748365d
        else return Integer.MIN_VALUE;  -214748365d
    }
    if (result == intAbsHigh) {
        if ((sign == 1) && (d > 7)) return Integer.MAX_VALUE; 2147483648
        if ((sign == -1) && (d > 8)) return Integer.MIN_VALUE; -2147483649
    }
    // 正常逻辑
    result = result * 10 + d;
    i++;
}
return sign*result;
}
```

```
// "  -42" 前导空格，符号位
// "4193abc" 后置非数字字符
// "abc987" 返回0
// "-9182838383838" 超过整数范围 返回Integer.MIN_VALUE
// “9182838383838” 超过整数范围 返回Integer.MAX_VALUE
// "" 空字符串 返回0
// "  " 全文空格 返回0
```

关注微信公众号“**小争哥**”，

后台回复“**PDF**”获取独家算法资料