```
                        chas[j--] = '2';
                        chas[j--] = '%';
                }
        }
}
```

补充问题。依然是从右向左倒着复制，遇到数字字符则直接复制，遇到“*”字符不复制。当把数字字符复制完，把左半区全部设置成“*”即可。具体请参看如下代码中的 modify 方法。

```
public void modify(char[] chas) {
        if (chas == null || chas.length == 0) {
                return;
        }
        int j = chas.length - 1;
        for (int i = chas.length - 1; i > -1; i--) {
                if (chas[i] != '*') {
                        chas[j--] = chas[i];
                }
        }
        for (; j > -1;) {
                chas[j--] = '*';
        }
}
```

以上两道题目都是利用倒着复制这个技巧，其实很多字符串问题也和这个小技巧有关。字符串的面试题一般不会太难，很多题目都是考查代码实现能力的。

翻转字符串

【题目】

给定一个字符类型的数组 chas，请在单词间做逆序调整。只要做到单词顺序逆序即可，对空格的位置没有特别要求。

【举例】

如果把 chas 看作字符串为"dog loves pig"，调整成"pig Loves dog"。

如果把 chas 看作字符串为"I’m a student."，调整成"student. a I'm"。

【补充题目】

给定一个字符类型的数组 chas 和一个整数 size，请把大小为 size 的左半区整体移到右半区，右半区整体移到左边。

【举例】

如果把 chas 看作字符串为"ABCDE"，size=3，调整成"DEABC"。

【要求】

如果 chas 长度为 N，两道题都要求时间复杂度为 $O(N)$，额外空间复杂度为 $O(1)$。

【难度】

士 ★☆☆☆

【解答】

原问题。首先把 chas 整体逆序。在逆序之后，遍历 chas 找到每一个单词，然后把每个单词里的字符逆序即可。比如“dog loves pig”，先整体逆序变为“gip sevol god”，然后每个单词进行逆序处理就变成了“pig loves dog”。逆序之后找每一个单词的逻辑，做到不出错即可。全部过程请参看如下代码中的 rotateWord 方法。

```
public void rotateWord(char[] chas) {
        if (chas == null || chas.length == 0) {
                return;
        }
        reverse(chas, 0, chas.length - 1);
        int l = -1;
        int r = -1;
        for (int i = 0; i < chas.length; i++) {
                if (chas[i] != ' ') {
                    l = i == 0 || chas[i - 1] == ' ' ? i : l;
                    r = i == chas.length - 1 || chas[i + 1] == ' ' ? i : r;
                }
                if (l != -1 && r != -1) {
                    reverse(chas, l, r);
                    l = -1;
                    r = -1;
                }
        }
}
```

```
public void reverse(char[] chas, int start, int end) {
        char tmp = 0;
        while (start < end) {
                tmp = chas[start];
                chas[start] = chas[end];
                chas[end] = tmp;
                start++;
                end--;
        }
}
```

补充问题，方法一。先把 chas[0..size-1]部分逆序，再把 chas[size..N-1]部分逆序，最后把 chas 整体逆序即可。比如，chas="ABCDE"，size=3。先把 chas[0..2]部分逆序，chas 变为"CBADE"，再把 chas[3..4]部分逆序，chas 变为"CBAED"，最后把 chas 整体逆序，chas 变为"DEABC"。具体过程请参看如下代码中的 rotate1 方法。

```
public static void rotate1(char[] chas, int size) {
        if (chas == null || size <= 0 || size >= chas.length) {
                return;
        }
        reverse(chas, 0, size - 1);
        reverse(chas, size, chas.length - 1);
        reverse(chas, 0, chas.length - 1);
}
```

方法二。用举例的方式来说明这个过程，chas="1234567ABCD"，size=7。

1．左部分为"1234567"，右部分为"ABCD"，右部分的长度为 4，比左部分小，所以把左部分前 4 个字符与右部分交换，chas[0..10]变为"ABCD5671234"。右部分小，所以右部分"ABCD"换过去再也不需要移动，剩下的部分为 chas[4..10]= "5671234"。左部分大，所以换过来的"1234"视为下一步的右部分，下一步的左部分为“567”。

2．左部分为"567"，右部分为"1234"，左部分的长度为 3，比右部分小，所以把右部分的后 3 个字符与左部分交换，chas[4..10]变为"2341567"。左部分小，所以左部分"567"换过去再也不需要移动，剩下的部分为 chas[4..7]= "2341"。右部分大，所以换过来的"234"视为下一步的左部分，下一步的右部分为"1"。

3．左部分为"234"，右部分为"1"。右部分的长度为 1，比左部分小，所以把左部分前 1 个字符与右部分交换，chas[4..7]变为"1342"。右部分小，所以右部分"1"换过去再也不需要移动，剩下的部分为 chas[5..7]= "342"。左部分大，所以换过来的"2"视为下一步的右部分，下一步的左部分为"34"。

4．左部分为"34"，右部分为"2"。右部分的长度为 1，比左部分小，所以把左部分前 1

个字符与右部分交换，chas[5..7]变为"243"。右部分小，所以右部分"2"换过去再也不需要移动，剩下的部分为 chas[6..7]= "43"。左部分大，所以换过来的"3"视为下一步的右部分，下一步的左部分为"4"。

5．左部分为"4"，右部分为"3"。一旦发现左部分跟右部分的长度一样，那么左部分和右部分完全交换即可，chas[6..7]变为"34"，整个过程结束，chas 已经变为"ABCD1234567"。

如果每一次左右部分的划分进行 M 次交换，那么都有 M 个字符再也不需要移动，而字符数一共为 N，所以交换行为最多发生 N 次。另外，如果某一次划分出的左右部分长度一样，那么交换完成后将不会再有新的划分，所以在很多时候交换行为会少于 N 次。比如，chas="1234ABCD"，size=4，最开始左部分为"1234"，右部分为"ABCD"，左右两个部分完全交换后为"ABCD1234"，同时不会有后续的划分，所以这种情况下一共只有 4 次交换行为。具体过程请参看如下代码中的 rotate2 方法。

```
public void rotate2(char[] chas, int size) {
        if (chas == null || size <= 0 || size >= chas.length) {
                return;
        }
        int start = 0;
        int end = chas.length - 1;
        int lpart = size;
        int rpart = chas.length - size;
        int s = Math.min(lpart, rpart);
        int d = lpart - rpart;
        while (true) {
                exchange(chas, start, end, s);
                if (d == 0) {
                        break;
                } else if (d > 0) {
                        start += s;
                        lpart = d;
                } else {
                        end -= s;
                        rpart = -d;
                }
                s = Math.min(lpart, rpart);
                d = lpart - rpart;
        }
}

public void exchange(char[] chas, int start, int end, int size) {
        int i = end - size + 1;
        char tmp = 0;
        while (size-- != 0) {
                tmp = chas[start];
                chas[start] = chas[i];
                chas[i] = tmp;
```

```
                start++;
                i++;
            }
        }
```

数组中两个字符串的最小距离

【题目】

给定一个字符串数组 strs，再给定两个字符串 str1 和 str2，返回在 strs 中 str1 与 str2 的最小距离，如果 str1 或 str2 为 null，或不在 strs 中，返回-1。

【举例】

strs=["1","3","3","3","2","3","1"]，str1="1"，str2="2"，返回 2。

strs=["CD"]，str1="CD"，str2="AB"，返回-1。

【进阶题目】

如果查询发生的次数有很多，如何把每次查询的时间复杂度降为 $O(1)$?

【难度】

尉 ★★☆☆

【解答】

原问题。从左到右遍历 strs，用变量 last1 记录最近一次出现的 str1 的位置，用变量 last2 记录最近一次出现的 str2 的位置。如果遍历到 str1，那么 i-last2 的值就是当前的 str1 和左边最它最近的 str2 之间的距离。如果遍历到 str2，那么 i-last1 的值就是当前的 str2 和左边最它最近的 str1 之间的距离。用变量 min 记录这些距离的最小值即可。请参看如下的 minDistance 方法。

```
public int minDistance(String[] strs, String str1, String str2) {
        if (str1 == null || str2 == null) {
                return -1;
        }
        if (str1.equals(str2)) {
                return 0;
        }
        int last1 = -1;
```