**http://blog.51cto.com/image/skin/artType01.jpg 字符串反转，神奇的算法-读《编程珠玑（第二版）》**

2011-05-06 15:06:57

标签：[算法](http://blog.51cto.com/tagindex.php?keyword=%CB%E3%B7%A8" \t "_blank) [读书笔记](http://blog.51cto.com/tagindex.php?keyword=%B6%C1%CA%E9%B1%CA%BC%C7) [休闲](http://blog.51cto.com/tagindex.php?keyword=%D0%DD%CF%D0) [字符串反转](http://blog.51cto.com/tagindex.php?keyword=%D7%D6%B7%FB%B4%AE%B7%B4%D7%AA) [编程珠玑](http://blog.51cto.com/tagindex.php?keyword=%B1%E0%B3%CC%D6%E9%E7%E1)

原创作品，允许转载，转载时请务必以超链接形式标明文章 [原始出处](http://haolloyin.blog.51cto.com/1177454/560597) 、作者信息和本声明。否则将追究法律责任。<http://haolloyin.blog.51cto.com/1177454/560597>

最近在缓慢地读《编程珠玑（第二版）》（英文名Programming Pearls)，书很薄（正文才160多页），但正如其封面“近20年来众多大师级程序员一致推崇的作品”所示，这本经典哪能是我一下子就能读完的？书中有很多简洁但有趣的例子分析，更有作者的实际经验及注意点的阐述，值得一点点地读，一遍不够以后再来第二、三遍...

另外，前些天发现51CTO读书频道有[《编程珠玑（续）》](http://book.51cto.com/art/201104/258062.htm" \l "book_content)(英文名 More Programming Pearls)整本书的在线阅读，真是太赞了！

回到正题。第二章“啊哈！算法”中作者给出三个“小题目”，其中第二个是：将一个具有n个元素的一维向量x向左旋转i个位置。例如，将 ABC123DEF456 向左旋转 3 个位置之后得到 123DEF456ABC。要求是花费与n成比例的时间来完成这个操作。

题目虽小，但正如本章标题“啊哈！算法”所示，作者想要强调的是算法的巧妙与强大，其中有本题的3种解决方案。

**\*.使用临时数组**

这大概是大多数人都会想到的一种方案，即将所给向量x的前i个复制到临时数组中，再将剩下的n-i 个元素往左移动i个位置，最后将临时数组中的i个元素复制回x中后面的位置。

代码实现：

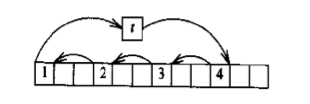
1. **public** **void** tempArrReverse(**char** [] arr, **int** len, **int** m){
2. **char** [] temp = **new** **char**[m];
3. // 将数组前m个元素保存到临时数组
4. **for**(**int** i = 0; i < m; i++){
5. temp[i] = arr[i];
6. }
7. // 将数组后面（len-m）个元素前移
8. **for**(**int** i = m; i < len; i++){
9. arr[i-m] = arr[i];
10. }
11. // 将临时数组所有元素复制到原数组
12. **for**(**int** i = 0; i < m; i++){
13. arr[len-m+i] = temp[i];
14. }
15. }

很明显上面的解法虽然简单移动，但用到了i个额外的空间，比较浪费空间。于是又有一个作者称赞“堪称巧妙的杂技表演”的解法。

**\*.“巧妙的杂技表演”**

先将x[0]移到临时变量t中，再把x[i]移到x[0]腾出来的位置中，x[2i]移到x[i]中，以此类推（每一次移动时将x[]中的下标对其长度n取模），直到又回到从x[0]提取元素，此时应该把临时变量t放入到上一个位置中，并开始下一次迭代，即对x[1]进行类似的操作。

找来书中直观的图如下：

[](http://img1.51cto.com/attachment/201105/150401110.png)

代码实现：

1. **public** **void** juggleReverse(**char** [] arr, **int** len, **int** m){
2. **int** k = 0;
3. **for**(**int** i = 0; i < m; i++){
4. k = 1;  //每次迭代都从1倍位移开始
5. **char** temp = arr[i];
6. **while**((k\*m+i) % len != i){
7. arr[(k-1)\*m+i] = arr[k\*m+i];
8. k++;
9. }
10. arr[(k-1)\*m+i] = temp;
11. }
12. }

但是，正如作者所提醒的“将这个想法转换为代码，请务必小心！”实际上我的实现中只能是当原向量长度n是位移i的整数倍时才可行，碰巧书中的也是用 i=3，n=12 这个例子来演示说明的，而且感觉此处的译文有点儿别扭，理解得不顺畅，得找找原文是怎么说的。

如果你在纸上画一下不是整数倍的情况，会发现其实情况复杂得多，有可能某些元素被放到其正确位置的后面去了，而我又不想用很多if...else语句来判断所出现的情况，因为我认为是我没理解到作者的思路，所以暂时只能这么实现（还要再思考），希望高手看到能指教一下。=)

**\*.另一种更巧妙的思路-原语的力量**

其实题目的意思就是字符串反转（reverse，不知道原文中“旋转”是不是这个词，得查证），书中这一小节“原语的力量”提到“有些编程语言提供了旋转作为对向量的原语运算”。假如已经有将数组指定部分元素进行反转的函数（原语操作），接下来的解法就更有意思了。

原问题可以这样来看：将原数组看成 ab，需要转换成 ba，先单独对子数组a进行反转得到a'b（a'表示a反转后的结果），同理单独反转b，得到 a'b',最后将得到的 a'b' 一起进行一次反转可得 （a'b'）',而这就是最终结果 ba了，不信你在纸上尝试一下（或者请看下面更有趣的、可操作的例子）。

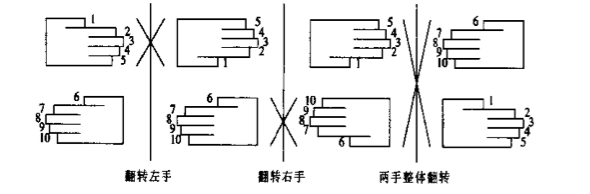
代码实现：

1. // 原语操作-反转数组
2. **public** **void** reverse(**char** [] arr, **int** start, **int** end){
3. **int** mid = (start + end) / 2;
4. **for**(**int** s = start, i = 1; i < mid; s++, i++){
5. **char** temp = arr[s];
6. arr[s] = arr[end-i];
7. arr[end-i] = temp;
8. }
9. }
11. **public** **void** useReverse(**char** [] arr, **int** len, **int** m){
12. reverse(arr, 0, m);
13. reverse(arr, m, len);
14. reverse(arr, 0, len);
15. }

其实到这里我认为作者已经让我感受到算法巧妙之处了，对于相同问题的不同理解、看法可以激发出这么精彩的解法。但实际上你还会发现这本经典书还很频繁地列举其他大师级人物对某问题的解法与运用之类的历史信息。例如，书中提到 Ken Thompson也是将这段代码用在其编辑器和转置代码的，而且“声称即使在那个时候，那也可编入神话故事了”。难道还不够精彩吗？请看下面。

**\*.可操作的证明方法-手摇法**

如果要将一个具有10个元素（我们只有10个手指啊）的数组向上旋转5个位置，先让两只手的掌心正对你自己，左右放在右手上面（其实两只手是同意平面上的），看下图：

[](http://img1.51cto.com/attachment/201105/150335209.png)

**其实如果亲自读这本书的话，会发现更多精彩！这样一本书，难道不值得一点点地读吗？**