结构之法 算法之道

Google/baidu搜“结构之法”，进入本博客；搜“为学论坛”，进入咱论坛。

个人资料

v_JULY_v

访问： 3782251次
积分： 25237分
排名： 第52名

原创： 133篇 转载： 0篇
译文： 5篇 评论： 9829条

博客公告

①.本blog开通于2010年10月11日，本blog算法交流群：149977547；北京程序员联盟：172727781。②.狂热算法，热爱数据挖掘，关注机器学习、统计分析，爱好文学数学。③.微博：研究者July，邮箱：zhoulei0907@yahoo.cn，欢迎诸君感恩资助@为学论坛，July，二零一三年二月八日。

我的微博

研究者July 北京

加关注

回复@泠雨冷月:我按照你的方法，平时意识到了就揉搓，可一个月过去了，发现心脏处还是如往常那般照样疼痛，看来，现在也不敢说自己还有一个健康的身体了，不必矫情，生死有命。//@泠雨冷月：应该是胸中积淤了，或者会有硬块，平时多揉搓，把那些地方按软了，血气通了就好~~~ 还有尽

文章分类

03.Algorithms（实现） (9)
01.Algorithms（研究） (27)
02.Algorithms（后续） (22)

程序员编程艺术第一章、左旋转字符串

分类： 08.MS 100' one Keys 11.TAOPP（编程艺术） 06.MS 100' answers 12.TAOPP string 2011-04-14 13:14 51328人阅读 评论(200) 收藏 举报

算法 编程 distance iterator string 面试

第一章、左旋转字符串

作者：July，yansha。
时间：二零一一年四月十四日。
微博：**http://weibo.com/julyweibo**。
出处：**http://blog.csdn.net/v_JULY_v**。

目录

前言

本人整理微软等公司面试100题系列，包括原题整理，资源上传，帖子维护，答案整理，勘误，修正与优化工作，包括后续全新整理的80道，总计180道面试题，已有半年的时间了。

关于这180道面试题的一切详情，请参见：**横空出世，席卷Csdn [评微软等数据结构+算法面试180题]**。

一直觉得，这180道题中的任何一题都值得自己反复思考，反复研究，不断修正，不断优化。之前的答案整理由于时间仓促，加之受最开始的认识局限，更兼水平有限，所以，这180道面试题的答案，有很多问题都值得进一步商榷与完善。

特此，想针对这180道面试题，再写一个系列，叫做：程序员编程艺术系列。如你所见，我一般确定要写成一个系列的东西，一般都会永久写下去的。

“他似风儿一般奔跑，很多人渐渐的停下来了，而只有他一直在飞，一直在飞....”

ok，本次程序员编程艺术系列以之前本人最初整理的微软面试100题中的第26题、**左旋转字符串，**为开篇，希望就此问题进行彻底而深入的阐述。然以下所有任何代码仅仅只是全部测试正确了而已，还有很多的优化工作要做。欢迎任何人，不吝赐教。谢谢。

第一节、左旋转字符串

题目描述：

定义字符串的左旋转操作：把字符串前面的若干个字符移动到字符串的尾部。
如把字符串abcdef左旋转2位得到字符串cdefab。
请实现字符串左旋转的函数，要求对长度为n的字符串操作的**时间复杂度为O(n)，空间复杂度为O(1)。**

编程之美上有这样一个类似的问题，咱们先来看一下：

> 设计一个算法，把一个含有N个元素的数组循环右移K位，要求时间复杂度为O（N），
> 且只允许使用两个附加变量。

分析：

我们先试验简单的办法，可以每次将数组中的元素右移一位，循环K次。
abcd1234→4abcd123→34abcd12→234abcd1→1234abcd。

```
RightShift(int* arr, int N, int K)
{
    while(K--)
    {
        int t = arr[N - 1];
        for(int i = N - 1; i > 0; i --)
            arr[i] = arr[i - 1];
        arr[0] = t;
    }
}
```

虽然这个算法可以实现数组的循环右移，但是算法复杂度为O（K * N），不符合题目的要求，要继续探索。

假如数组为abcd1234，循环右移4位的话，我们希望到达的状态是1234abcd。
不妨设K是一个非负的整数，当K为负整数的时候，右移K位，相当于左移（-K）位。
左移和右移在本质上是一样的。

解法一：
大家开始可能会有这样的潜在假设，K<N。事实上，很多时候也的确是这样的。但严格来说，我们不能用这样的“惯性思维”来思考问题。
尤其在编程的时候，全面地考虑问题是很重要的，K可能是一个远大于N的整数，在这个时候，上面的解法是需要改进的。
仔细观察循环右移的特点，不难发现：每个元素右移N位后都会回到自己的位置上。因此，如果K > N，右移K-N之后的数组序列跟右移K位的结果是一样的。

进而可得出一条通用的规律：
右移K位之后的情形，跟右移K'= K % N位之后的情形一样，如代码清单2-34所示。

```
//代码清单2-34
RightShift(int* arr, int N, int K)
{
    K %= N;
    while(K--)
    {
        int t = arr[N - 1];
        for(int i = N - 1; i > 0; i --)
            arr[i] = arr[i - 1];
        arr[0] = t;
    }
}
```

可见，增加考虑循环右移的特点之后，算法复杂度降为O（N^2），这跟K无关，与题目的要求又接近了一步。但时间复杂度还不够低，接下来让我们继续挖掘循环右移前后，数组之间的关联。

解法二：

假设原数组序列为abcd1234，要求变换成的数组序列为1234abcd，即循环右移了4位。比较之后，不难看出，其中有两段的顺序是不变的：1234和abcd，可把这两段看成两个整体。右移K位的过程就是把数组的两部分交换一下。

变换的过程通过以下步骤完成：

逆序排列abcd：abcd1234 → dcba1234；

逆序排列1234：dcba1234 → dcba4321；

全部逆序：dcba4321 → 1234abcd。

伪代码可以参考清单2-35。

```
//代码清单2-35
Reverse(int* arr, int b, int e)
{
    for(; b < e; b++, e--)
    {
        int temp = arr[e];
        arr[e] = arr[b];
        arr[b] = temp;
    }
}

RightShift(int* arr, int N, int k)
{
    K %= N;
    Reverse(arr, 0, N – K - 1);
    Reverse(arr, N - K, N - 1);
    Reverse(arr, 0, N - 1);
}
```

这样，我们就可以在线性时间内实现右移操作了。

稍微总结下：

编程之美上，

（限制书中思路的根本原因是，题目要求：“且只允许使用两个附加变量”，去掉这个限制，思路便可如泉喷涌）

1、第一个想法，是一个字符一个字符的右移，所以，复杂度为O（N*K）

2、后来，它改进了，通过这条规律：右移K位之后的情形，跟右移K'= K % N位之后的情形一样

复杂度为O（N^2）

3、直到最后，它才提出三次翻转的算法，得到线性复杂度。

下面，你将看到，本章里我们的做法是：

1、三次翻转，直接线性

2、两个指针逐步翻转，线性

3、stl的rotate算法，线性

好的，现在，回到咱们的左旋转字符串的问题中来，对于这个左旋转字符串的问题，咱们可以如下这样考虑：

1.1、思路一：

对于这个问题，咱们换一个角度，可以这么做：

将一个字符串分成两部分，X和Y两个部分，在字符串上定义反转的操作X^T，即把X的所有字符反转（如，X="abc"，那么X^T="cba"），那么我们可以得到下面的结论：(X^TY^T)^T=YX。显然我们这就可以转化为字符串的反转的问题了。

不是么?ok,就拿abcdef 这个例子来说（非常简短的三句，请细看，一看就懂）：

1、首先分为俩部分，X:abc，Y:def；

2、X->X^T，abc->cba， Y->Y^T，def->fed。

3、(X^TY^T)^T=YX，cbafed->defabc，即整个翻转。

我想，这下，你应该了然了。

然后，代码可以这么写（已测试正确）：

```
//Copyright@ 小桥流水 && July
//c代码实现，已测试正确。
//http://www.smallbridge.co.cc/2011/03/13/100%E9%A2%98
//_21-%E5%B7%A6%E6%97%8B%E8%BD%AC%E5%AD%97%E7%AC%A6%E4%B8%B2.html
//July、updated，2011.04.17。
#include <stdio.h>
#include <string.h>

char * invert(char *start, char *end)
{
    char tmp, *ptmp = start;
    while (start != NULL && end != NULL && start < end)
    {
        tmp = *start;
        *start = *end;
        *end = tmp;
        start ++;
        end --;
    }
    return ptmp;
}

char *left(char *s, int pos)    //pos为要旋转的字符个数，或长度，下面主函数测试中，pos=3。
{
    int len = strlen(s);
    invert(s, s + (pos - 1));  //如上，X->X^T，即 abc->cba
    invert(s + pos, s + (len - 1)); //如上，Y->Y^T，即 def->fed
    invert(s, s + (len - 1));  //如上，整个翻转，(X^TY^T)^T=YX，即 cbafed->defabc。
    return s;
}

int main()
{
    char s[] = "abcdefghij";
    puts(left(s, 3));
    return 0;
}
```

1.2、答案V0.3版中，第26题勘误：

之前的答案V0.3版[第21-40题答案]中，第26题、贴的答案有误，那段代码的问题，最早是被网友Sorehead给指出来的：

> 第二十六题：
> 楼主的思路确实很巧妙，我真没想到还有这种方法，学习了。
> 不过楼主代码中存在问题，主要是条件判断部分：
> 函数LeftRotateString中 if (nLength > 0 || n == 0 || n > nLength)
> 函数ReverseString中 if (pStart == NULL || pEnd == NULL)

当时，以答案整理因时间仓促，及最开始考虑问题不够周全为由，没有深入细看下去。后来，朋友达摩流浪者再次指出了上述代码的问题：

> 26题 这句 if(nLength > 0 || n == 0 || n > nLength)，有问题吧?
> 还有一句，应该是if(!(pStart == NULL || pEnd == NULL))，吧。

而后，修改如下（已测试正确）

```
//zhedahht
//July、k，updated
//copyright @2011.04.14，by July。
//引用，请注明原作者，出处。
#include <string.h>
#include <iostream>
using namespace std;

void Swap(char* a,char* b)  //特此把交换函数，独立抽取出来。当然，不排除会有人认为，此为多此一举。
{
    char temp =*a;
    *a = *b;
    *b = temp;
}

// Reverse the string between pStart and pEnd
void ReverseString(char* pStart, char* pEnd)
{
    if(*pStart != '/0' && *pEnd != '/0')
        //这句也可以是：if(pStart != NULL && pEnd != NULL)。
    {
        while(pStart <= pEnd)
        {
            Swap(pStart,pEnd);   //交换

            pStart++;
            pEnd--;
```

```
        }
    }
}

// Move the first n chars in a string to its end
char* LeftRotateString(char* pStr, unsigned int n)
{
    if(pStr != NULL)
    {
        int nLength = static_cast<int>(strlen(pStr));
        if(nLength >0 && n != 0 && n<nLength)    //n可以=0,也可以说不该=0。
            //nLength是整个字符串的长度，n是左边的一部分，所以应该是n<nLength。
            //之前上传的答案（代码），就错在这里，最初的为n>nLength,当然，就是错的了。July、
k, updated。
        {
            char* pFirstStart = pStr;
            char* pFirstEnd = pStr + n - 1;
            char* pSecondStart = pStr + n;
            char* pSecondEnd = pStr + nLength - 1;

            // reverse the first part of the string
            ReverseString(pFirstStart, pFirstEnd);
            // reverse the second part of the strint
            ReverseString(pSecondStart, pSecondEnd);
            // reverse the whole string
            ReverseString(pFirstStart, pSecondEnd);
        }
    }
    return pStr;
}

int main()
{
    char a[11]="hello July";    //2、修正，以一个数组实现存储整个字符串
    char *ps=a;
    LeftRotateString(ps, 6);
    for(;*ps!='/0';ps++)
        cout<<*ps;
    cout<<endl;
    ps=NULL;    //代码规范
    return 0;
}
```

上述，修正的俩处错误，如下所示：

1、如上注释中所述：

if(nLength >0 && n<nLength)

//nLength是整个字符串的长度吧，n是左边的一部分，所以应该是n<nLength。

2、至于之前的主函数为什么编写错误，请看下面的注释：

```
int main()
{
    char *ps="hello July";  //本身没错，但你不能对ps所指的字符串做任何修改。
    //这句其实等价于：const char *ps = "hello July"
    LeftShiftString( ps, 4 );  //而在这里，试图修改ps所指的字符串常量，所以将出现错误。
    puts( ps );
    return 0;
}
```

当然，上面的解释也不是完全正确的，正如ivan所说：从编程实践来说，不完全对。

如果在一个大的工程里面，你怎么知道ps指向的是""字符串，还是malloc出来的东西？

那么如何决定要不要对ps进行delete?

不过，至少第26题的思路一的代码，最终完整修正完全了。

1.3、updated:

可能你还是感觉上述代码，有点不好理解，ok，下面再给出一段**c实现的代码**。

然后，我们可以看到c的高效与简洁。

```
//copyright@ yiyibupt&&July
//已测试正确，July、updated，2011.04.17.
//不要小看每一段程序，July。
#include <cstdio>
#include <cstring>

void rotate(char *start, char *end)
{
    while(start != NULL && end !=NULL && start<end)
    {
        char temp=*start;
        *start=*end;
        *end=temp;
        start++;
        end--;
    }
```

```
}

void leftrotate(char *p,int m)
{
    if(p==NULL)
        return ;
    int len=strlen(p);
    if(m>0&&m<=len)
    {
        char *xfirst,*xend;
        char *yfirst,*yend;
        xfirst=p;
        xend=p+m-1;
        yfirst=p+m;
        yend=p+len-1;
        rotate(xfirst,xend);
        rotate(yfirst,yend);
        rotate(p,p+len-1);
    }
}

int main(void)
{
    char str[]="abcdefghij";
    leftrotate(str,3);
    printf("%s/n",str);
    return 0;
}
```

第二节、两指针逐步翻转

先看下网友litaoye 的回复：26.左旋转字符串跟panda所想，是一样的，即，

以abcdef为例

1. ab->ba

2. cdef->fedc

原字符串变为bafedc

3. 整个翻转：cdefab

//只要俩次翻转，且时间复杂度也为O（n）。

2.1、在此，本人再奉献另外一种思路，即为本思路二：

abc defghi，要abc移动至最后

abc defghi->def abcghi->def ghiabc

定义俩指针，p1指向ch[0]，p2指向ch[m]；

一下过程循环m次，交换p1和p2所指元素，然后p1++, p2++；。

第一步，交换abc 和def，

abc defghi->def abcghi

第二步，交换abc 和 ghi，

def abcghi->def ghiabc

整个过程，看起来，就是abc 一步一步 向后移动

abc defghi

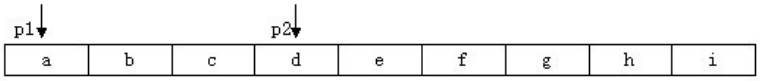
def abcghi

def ghi abc

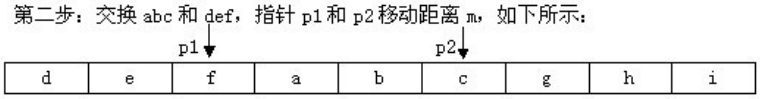
//最后的 复杂度是O（m+n）

以下是朋友颜沙针对上述过程给出的图解：

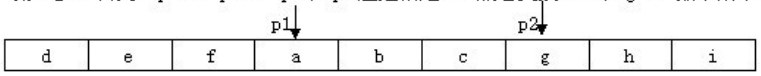
第一步：指针处于初始位置，如下所示：

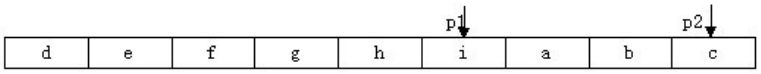


第二步：交换 abc 和 def，指针 p1 和 p2 移动距离 m，如下所示：



第三步：首先，p1++，p2++，p1 和 p2 还是相距 m，然后交换 abc 和 ghi，如下所示：





至此，整个过程结束，得到最终结果 defghi abc。

2.2、各位读者注意了：

由上述例子九个元素的序列abcdefghi，您已经看到，m=3时，p2恰好指到了数组最后一个元素，于是，上述思路没有问题。但如果上面例子中i 的后面还有元素列?

即，如果是要左旋十个元素的序列：abcdefghij，ok，下面，就举这个例子，对abcdefghij序列进行左旋转操作：

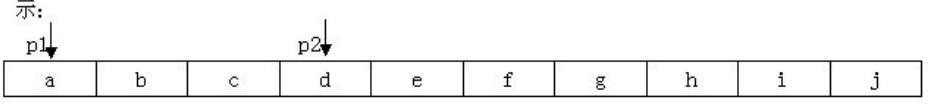
如果abcdef ghij要变成defghij abc：

abcdef ghij

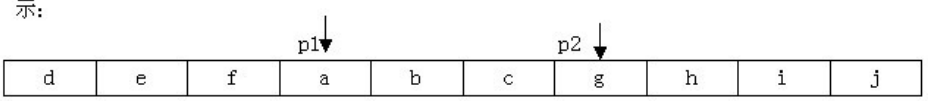
1. def abc ghij
2. def ghi abc j　　//接下来，j 步步前移
3. def ghi ab jc
4. def ghi a j bc
5. def ghi j abc

下面，再针对上述过程，画个图清晰说明下，如下所示：

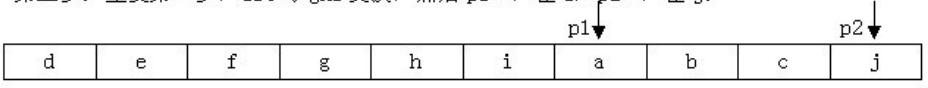
第一步：指针处于初始位置，其中 p1 指向首地址，p2 指向 p1+m（本例 m 为 3），如下图所示：



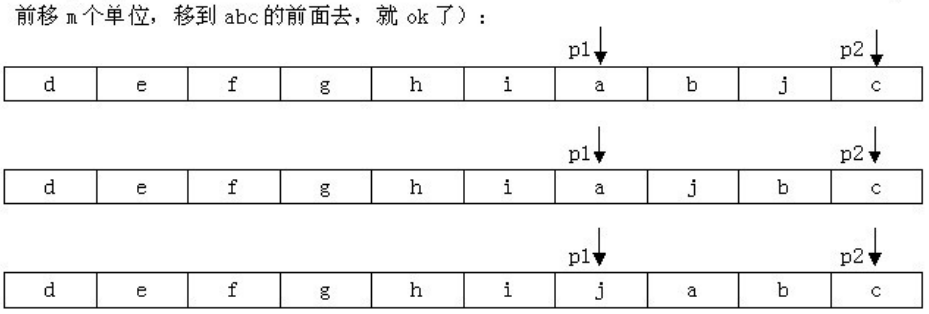
第二步：交换 p1 和 p2 所指元素，循环 m 次，abc 与 def 交换，然后 p1++, p2++，如下图所示：



第三步：重复第二步，abc 与 ghi 交换，然后 p1++，在 a，p2++，在 j：



第四步：如果p2+m-1不越界，说明p2到数组末尾之间所包含的元素为m，即为上例（指abcdefghi，九个元素）讨论的情况。否则，说明p2到数组末尾之间所包含的元素小于m，保持p1，p2不变，将这些元素向左移动m个单元即可，如下图所示（下图的情况，就是j前移m个单位，移到abc的前面去，就ok了）：



至此，整个过程结束，得到最终结果。

ok，咱们来好好彻底总结一下此思路二：（就4点，请仔细阅读）：

1、首先让p1=ch[0]，p2=ch[m]，即让p1，p2相隔m的距离；

2、判断p2+m-1是否越界，如果没有越界转到3，否则转到4（abcdefgh这8个字母的字符串，以4左旋，那么初始时p2指向e，p2+4越界了，但事实上p2至p2+m-1是m个字符，可以再做一个交换）。

3、不断交换*p1与*p2，然后p1++，p2++，循环m次，然后转到2。

4、此时p2+m-1 已经越界，在此只需处理尾巴。过程如下：

4.1 通过n-p2得到p2与尾部之间元素个数r，即我们要前移的元素个数。

4.2 以下过程执行r次：

ch[p2]<->ch[p2-1]，ch[p2-1]<->ch[p2-2]，....，ch[p1+1]<->ch[p1]；p1++；p2++；

（特别感谢tctop组成员big的指正，tctop组的修订wiki页面为：**http://tctop.wikispaces.com/**）

所以，之前最初的那个左旋转九个元素abcdefghi的思路在末尾会出现问题的（如果p2后面有元素就不能这么变，例如，如果是处理十个元素，abcdefghij 列?对的，就是这个意思），解决办法有两个：

方法一（即如上述思路总结所述）：

def ghi abc jk

当p1指向a，p2指向j时，由于p2+m越界，那么此时p1，p2不要变

这里p1之后（abcjk)就是尾巴，处理尾巴只需将j,k移到abc之前，得到最终序列，代码编写如下：

```
//copyright@July、颜沙
//最终代码，July，updated again，2011.04.17。
#include <iostream>
#include <string>
using namespace std;

void rotate(string &str, int m)
{

    if (str.length() == 0 || m <= 0)
        return;

    int n = str.length();

    if (m % n <= 0)
        return;

    int p1 = 0, p2 = m;
    int k = (n - m) - n % m;

    // 交换p1，p2指向的元素，然后移动p1，p2
    while (k --)
    {
        swap(str[p1], str[p2]);
        p1++;
        p2++;
    }

    // 重点，都在下述几行。
    // 处理尾部，r为尾部左移次数
    int r = n - p2;
    while (r--)
    {
        int i = p2;
        while (i > p1)
        {
            swap(str[i], str[i-1]);
            i--;
        }
        p2++;
        p1++;
    }
    //比如一个例子，abcdefghijk
```

```
    //                      p1    p2
    //当执行到这里时，defghi a b c j k
    //p2+m出界 了，
    //r=n-p2=2，所以以下过程，要执行循环俩次。

    //第一次：j 步步前移，abcjk->abjck->ajbck->jabck
    //然后，p1++，p2++，p1指a，p2指k。
    //                  p1    p2
    //第二次：defghi j a b c k
    //同理，此后，k步步前移，abck->abkc->akbc->kabc。
}

int main()
{
    string ch="abcdefghijk";
    rotate(ch,3);
    cout<<ch<<endl;
    return 0;
}
```

方法二：
def ghi abc jk
当p1指向a，p2指向j时，那么交换p1和p2，
此时为:
def ghi jbc ak
p1++，p2++,p1指向b，p2指向k，继续上面步骤得:
def ghi jkc ab
p1++，p2不动,p1指向c，p2指向b，p1和p2之间（cab)也就是尾巴，
那么处理尾巴（cab)需要循环左移一定次数（而后的具体操作步骤已在下述程序的注释中已详细给出）。

根据方案二，不难写出下述代码（已测试正确）：

```
#include <iostream>
#include <string>
using namespace std;

//颜沙，思路二之方案二，
//July、updated，2011.04.16。
void rotate(string &str, int m)
{
    if (str.length() == 0 || m < 0)
        return;

    //初始化p1，p2
    int p1 = 0, p2 = m;
    int n = str.length();

    // 处理m大于n
    if (m % n == 0)
        return;

    // 循环直至p2到达字符串末尾
    while(true)
    {
        swap(str[p1], str[p2]);
        p1++;
        if (p2 < n - 1)
            p2++;
        else
            break;
    }

    // 处理尾部，r为尾部循环左移次数
    int r = m - n % m;  // r = 1.
    while (r--)  //外循环执行一次
    {
        int i = p1;
        char temp = str[p1];
        while (i < p2)  //内循环执行俩次
        {
            str[i] = str[i+1];
            i++;
        }
        str[p2] = temp;
    }
    //举一个例子
    //abcdefghijk
    //当执行到这里的时候，defghiabcjk
    //      p1    p2
    //defghi a b c j k，a 与 j交换，jbcak，然后，p1++，p2++
    //        p1    p2
    //       j b c a k，b 与 k交换，jkcab，然后，p1++，p2不动，

    //r = m - n % m= 3-11%3=1，即循环移位1次。
    //          p1  p2
    //       j k c a b
    //p1所指元素c实现保存在temp里，
    //然后执行此条语句：str[i] = str[i+1]; 即a跑到c的位置处，a_b
    //i++，再次执行：str[i] = str[i+1]，ab
    //最后，保存好的c 填入，为abc，所以，最终序列为defghi jk abc。
    //July、updated，2011.04.17晚，送走了她。
}

int main()
{
    string ch="abcdefghijk";
    rotate(ch,3);
    cout<<ch<<endl;
    return 0;
}
```

注意：上文中都是假设m<n，且如果鲁棒点的话令m=m%n，这样m允许大于n。另外，各位要记得处理指针为空的情况。

还可以看下这段代码：

```
/*
 * myinvert2.cpp
 *
 *  Created on: 2011-5-11
 *      Author: BigPotato
 */
#include<iostream>
#include<string>
#define positiveMod(m,n) ((m) % (n) + (n)) % (n)

/*
 *左旋字符串str，m为负数时表示右旋abs(m)个字母
 */
void rotate(std::string &str, int m) {
    if (str.length() == 0)
        return;
    int n = str.length();
    //处理大于str长度及m为负数的情况,positiveMod可以取得m为负数时对n取余得到正数
    m = positiveMod(m,n);
    if (m == 0)
        return;
    //    if (m % n <= 0)
    //        return;
    int p1 = 0, p2 = m;
    int round;
    //p2当前所指和之后的m-1个字母共m个字母，就可以和p2前面的m个字母交换。
    while (p2 + m - 1 < n) {
        round = m;
        while (round--) {
            std::swap(str[p1], str[p2]);
            p1++;
            p2++;
        }
    }
    //剩下的不足m个字母逐个交换
    int r = n - p2;
    while (r--) {
        int i = p2;
        while (i > p1) {
            std::swap(str[i], str[i - 1]);
            i--;
        }
        p2++;
        p1++;
    }
}

//测试
int main(int argc, char **argv) {
    //    std::cout << ((-15) % 7 + 7) % 7 << std::endl;
    //    std::cout << (-15) % 7 << std::endl;
    std::string ch = "abcdefg";
    int len = ch.length();
    for (int m = -2 * len; m <= len * 2; m++) {
        //由于传给rotate的是string的引用，所以这里每次调用都用了一个新的字符串
        std::string s = "abcdefg";
        rotate(s, m);
        std::cout << positiveMod(m,len) << ": " << s << std::endl;
    }

    return 0;
}
```

第三节、通过递归转换，缩小问题之规模

本文最初发布时，网友留言bluesmic说：楼主，谢谢你提出的研讨主题，很有学术和实践价值。关于思路二，本人提一个建议：思路二的代码，如果用递归的思想去简化，无论代码还是逻辑都会更加简单明了。

就是说，把一个规模为N的问题化解为规模为M(M<N)的问题。

举例来说，设字符串总长度为L，左侧要旋转的部分长度为s1，那么当从左向右循环交换长度为s1的小段，直到最后，由于剩余的部分长度为s2(s2==L%s1)而不能直接交换。

该问题可以递归转化成规模为s1+s2的，方向相反(从右向左)的同一个问题。随着递归的进行，左右反复回荡，直到某一次满足条件L%s1==0而交换结束。

举例解释一下：

设原始问题为：将“123abcdefg”左旋转为“abcdefg123”，即总长度为10，旋转部("123")长度为3的左旋转。按照思路二的运算，演变过程为“123abcdefg”->"abc123defg"->"abcdef123g"。这时，"123"无法和"g"作对调，该问题递归转化为：将“123g”右旋转为"g123"，即总长度为4，旋转部("g")长度为1的右旋转。

updated：

Ys：

Bluesmic的思路没有问题，他的思路以前很少有人提出。思路是通过递归将问题规模变小。当字符串总长度为n，左侧要旋转的部分长度为m，那么当从左向右循环交换长度为m的小段直到剩余部分为m'(n % m)，此时m' < m，已不能直接交换了。

此后，我们换一个思路，把该问题递归转化成规模大小为m' +m，方向相反的同一问题。随着递归的进行，直到满足结束条件n % m==0。

举个具体事例说明，如下：

1、对于字符串abc def ghi gk，

将abc右移到def ghi gk后面，此时n = 11，m = 3，m' = n % m = 2;

abc def ghi gk -> def ghi abc gk

2、问题变成gk左移到abc前面，此时n = m' + m = 5，m = 2，m' = n % m 1;

abc gk -> a gk bc

3、问题变成a右移到gk后面，此时n = m' + m = 3，m = 1，m' = n % m = 0;

a gk bc-> gk a bc。 由于此刻，n % m = 0，满足结束条件，返回结果。

即从左至右，后从右至左，再从左至右，如此反反复复，直到满足条件，返回退出。

代码如下，已测试正确（有待优化）：

```
//递归，
//感谢网友Bluesmic提供的思路

//copyright@ yansha 2011.04.19
//July, updated, 2011.04.20.
#include <iostream>
using namespace std;

void rotate(string &str, int n, int m, int head, int tail, bool flag)
{
    //n 待处理部分的字符串长度，m：待处理部分的旋转长度
    //head：待处理部分的头指针，tail：待处理部分的尾指针
    //flag = true进行左旋，flag = false进行右旋

    // 返回条件
    if (head == tail || m <= 0)
        return;

    if (flag == true)
    {
        int p1 = head;
        int p2 = head + m;  //初始化p1，p2

        //1、左旋：对于字符串abc def ghi gk，
        //将abc右移到def ghi gk后面，此时n = 11，m = 3，m' = n % m = 2;
        //abc def ghi gk -> def ghi abc gk
        //（相信，经过上文中那么多繁杂的叙述，此类的转换过程，你应该是了如指掌了。）

        int k = (n - m) - n % m;    //p1，p2移动距离，向右移六步

        /*----------------------
        解释下上面的k = (n - m) - n % m的由来：
        yansha：
        以p2为移动的参照系：
        n-m 是开始时p2到末尾的长度，n%m是尾巴长度
        (n-m)-n%m就是p2移动的距离
        比如 abc def efg hi
        开始时p2->d,那么n-m 为def efg hi的长度8，
        n%m 为尾巴hi的长度2，
        因为我知道abc要移动到hi的前面，所以移动长度是
        (n-m)-n%m = 8-2 = 6。
        */

        for (int i = 0; i < k; i++, p1++, p2++)
            swap(str[p1], str[p2]);

        rotate(str, n - k, n % m, p1, tail, false);  //flag标志变为false，结束左旋，下面，进入右旋
    }
    else
    {
        //2、右旋：问题变成gk左移到abc前面，此时n = m' + m = 5，m = 2，m' = n % m 1;
        //abc gk -> a gk bc

        int p1 = tail;
        int p2 = tail - m;

        // p1，p2移动距离，向左移俩步
        int k = (n - m) - n % m;

        for (int i = 0; i < k; i++, p1--, p2--)
            swap(str[p1], str[p2]);
```

```
        rotate(str, n - k, n % m, head, p1, true);  //再次进入上面的左旋部分，
        //3、左旋：问题变成a右移到gk后面，此时n = m' + m = 3, m = 1, m' = n % m = 0;
        //a gk bc-> gk a bc。 由于此刻，n % m = 0，满足结束条件，返回结果。

    }
}

int main()
{
    int i=3;
    string str = "abcdefghijk";
    int len = str.length();
    rotate(str, len, i % len, 0, len - 1, true);
    cout << str.c_str() << endl;   //转化成字符数组的形式输出
    return 0;
}
```

非常感谢。

稍后，由下文，您将看到，其实上述思路二的本质即是**下文将要阐述的stl rotate算法**，详情，请继续往下阅读。

第四节、stl::rotate 算法的步步深入

思路三：

3.1、数组循环移位

下面，我将再具体深入阐述下此STL 里的rotate算法，由于stl里的rotate算法，用到了gcd的原理，下面，我将**先介绍此辗转相除法，或欧几里得算法，gcd的算法思路及原理。**

gcd，即辗转相除法，又称欧几里得算法，是求最大公约数的算法，即求两个正整数之最大公因子的算法。此算法作为TAOCP第一个算法被阐述，足见此算法被重视的程度。

gcd算法：给定俩个正整数m，n（m>=n），求它们的最大公约数。（注意，一般要求m>=n，若m<n，则要先交换m<->n。下文，会具体解释）。以下，是此算法的具体流程：

1、[求余数]，令r=m%n，r为n除m所得余数（0<=r<n）；

2、[余数为0?]，若r=0，算法结束，此刻，n即为所求答案，否则，继续，转到3；

3、[重置]，置m<-n，n<-r，返回步骤1.

此算法的证明，可参考计算机程序设计艺术第一卷：基本算法。证明，此处略。

ok，下面，举一个例子，你可能看的更明朗点。

比如，给定m=544，n=119，

则余数r=m%n=544%119=68; 因r!=0，所以跳过上述步骤2，执行步骤3。；

置m<-119，n<-68，=>r=m%n=119%68=51;

置m<-68，n<-51，=>r=m%n=68%51=17；

置m<-51，n<-17，=>r=m%n=51%17=0，算法结束，

此时的n=17，即为m=544，n=119所求的俩个数的最大公约数。

再解释下上述gcd(m，n)算法开头处的，要求m>=n 的原因：举这样一个例子，如m<n，即m=119，n=544的话，那么r=m%n=119%544=119,

因为r!=0,所以执行上述步骤3，注意，看清楚了：m<-544，n<-119。看到了没，尽管刚开始给的m<n，但最终执行gcd算法时，还是会把m，n的值交换过来，以保证m>=n。

ok，我想，现在，你已经彻底明白了此gcd算法，下面，咱们进入主题，stl里的rotate算法的具体实现。//待续。

熟悉stl里的rotate算法的人知道，对长度为n的数组(ab)左移m位，可以用stl的rotate函数（stl针对三种不同的迭代器，提供了三个版本的rotate）。但在某些情况下，用stl的rotate效率极差。

对数组循环移位，可以采用的方法有（**也算是对上文思路一，和思路二的总结**）：

flyinghearts：

① 动态分配一个同样长度的数组，将数据复制到该数组并改变次序，再复制回原数组。（最

普通的方法）

② 利用ba=(br)^T(ar)^T=(arbr)^T，通过三次反转字符串。（即上述思路一，首先对序列前部分逆序，再对序列后部分逆序，再对整个序列全部逆序）

③ 分组交换（尽可能使数组的前面连续几个数为所要结果）：

若a长度大于b，将ab分成a0a1b，交换a0和b，得ba1a0，只需再交换a1 和a0。

若a长度小于b，将ab分成ab0b1，交换a和b0，得b0ab1，只需再交换a 和b0。

通过不断将数组划分，和交换，直到不能再划分为止。分组过程与求最大公约数很相似。

④ 所有序号为 (j+i *m) % n (j 表示每个循环链起始位置，i 为计数变量，m表示左旋转位数，n表示字符串长度)，会构成一个循环链（共有gcd(n,m)个，gcd为n、m的最大公约数），每个循环链上的元素只要移动一个位置即可，最后整个过程总共交换了n次（每一次循环链，是交换n/gcd(n,m)次，总共gcd(n,m)个循环链。所以，总共交换n次）。

stl的rotate的三种迭代器，即是，分别采用了后三种方法。

在给出stl rotate的源码之前，先来看下我的朋友ys对上述第④ 种方法的评论：

ys：这条思路个人认为绝妙，也正好说明了数学对算法的重要影响。

通过前面思路的阐述，我们知道对于循环移位，最重要的是指针所指单元不能重复。例如要使abcd循环移位变成dabc(这里m=3,n=4)，经过以下一系列眼花缭乱的赋值过程就可以实现：

ch[0]->temp, ch[3]->ch[0], ch[2]->ch[3], ch[1]->ch[2], temp->ch[1]; （*）

字符串变化为：abcd->_bcd->dbc_->db_c->d_bc->dabc;

是不是很神奇？其实这是有规律可循的。

请先看下面的说明再回过头来看。

对于左旋转字符串，我们知道每个单元都需要且只需要赋值一次，什么样的序列能保证每个单元都只赋值一次呢？

1、对于正整数m、n互为质数的情况，通过以下过程得到序列的满足上面的要求：

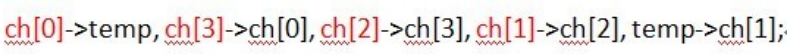
```
for i = 0: n-1
    k = i * m % n;
end
```

举个例子来说明一下，例如对于m=3,n=4的情况，

1、我们得到的序列：即通过上述式子求出来的k序列，是0, 3, 2, 1。

2、然后，你只要只需按这个顺序赋值一遍就达到左旋3的目的了：

ch[0]->temp, ch[3]->ch[0], ch[2]->ch[3], ch[1]->ch[2], temp->ch[1]; （*）



ok，这是不是就是按上面（*）式子的顺序所依次赋值的序列阿?哈哈，很巧妙吧。当然，以上只是特例，作为一个循环链，相当于rotate算法的一次内循环。

2、对于正整数m、n不是互为质数的情况（因为不可能所有的m，n都是互质整数对），那么我们把它分成一个个互不影响的循环链，正如flyinghearts所言，所有序号为 (j + i * m) % n（j为0到gcd(n, m)-1之间的某一整数，i = 0:n-1）会构成一个循环链，一共有gcd(n, m)个循环链，对每个循环链分别进行一次内循环就行了。

综合上述两种情况，可简单编写代码如下：

```
//④ 所有序号为 (j+i *m) % n (j 表示每个循环链起始位置，i 为计数变量，m表示左旋转位数，n表示字符串长度)，
//会构成一个循环链（共有gcd(n,m)个，gcd为n、m的最大公约数），

//每个循环链上的元素只要移动一个位置即可，最后整个过程总共交换了n次
//（每一次循环链，是交换n/gcd(n,m)次，共有gcd(n,m)个循环链，所以，总共交换n次）。

void rotate(string &str, int m)
{
    int lenOfStr = str.length();
    int numOfGroup = gcd(lenOfStr, m);
    int elemInSub = lenOfStr / numOfGroup;

    for(int j = 0; j < numOfGroup; j++)
        //对应上面的文字描述，外循环次数j为循环链的个数，即gcd(n, m)个循环链
    {
        char tmp = str[j];

        for (int i = 0; i < elemInSub - 1; i++)
            //内循环次数i为，每个循环链上的元素个数，n/gcd(m,n)次
            str[(j + i * m) % lenOfStr] = str[(j + (i + 1) * m) % lenOfStr];
```

```
        str[(j + i * m) % lenOfStr] = tmp;
    }
}
```

后来有网友针对上述的思路④，给出了下述的证明：

1、首先，直观的看肯定是有循环链，关键是有几条以及每条有多长，根据(i+j *m) % n这个表达式可以推出一些东东，一个j对应一条循环链，现在要证明(i+j *m) % n有n/gcd(n,m)个不同的数。

2、假设j和k对应的数字是相同的， 即(i+j*m)%n = (i+k*m)%n， 可以推出n|(j-k)*m，m=m'*gcd(n.m), n=n'*gcd(n,m), 可以推出n'|(j-k)*m'，而m'和n'互素，于是n'|(j-k)，即(n/gcd(n,m))|(j-k)，

3、所以(i+j*m) % n有n/gcd(n,m)个不同的数。则总共有gcd(n，m)个循环链。符号"|"是整除的意思。

以上的3点关于为什么一共有gcd(n, m)个循环链的证明，应该是来自qq3128739xx的，非常感谢这位朋友。

3.2、以下，便是摘自sgi stl v3.3版中的stl_algo_h文件里，有关rotate的实现的代码：

```
// rotate and rotate_copy, and their auxiliary functions
template <class _EuclideanRingElement>
_EuclideanRingElement __gcd(_EuclideanRingElement __m,
                            _EuclideanRingElement __n)
{  //gcd(m,n)实现
    while (__n != 0) {
        _EuclideanRingElement __t = __m % __n;
        __m = __n;
        __n = __t;
    }
    return __m;   //....
}

//③ 分组交换（尽可能使数组的前面连续几个数为所要结果）：
//若a长度大于b，将ab分成a0a1b，交换a0和b，得ba1a0，只需再交换a1 和a0。
//若a长度小于b，将ab分成ab0b1，交换a和b0，得b0ab1，只需再交换a 和b0。
//通过不断将数组划分，和交换，直到不能再划分为止。分组过程与求最大公约数很相似。
template <class _ForwardIter, class _Distance>
_ForwardIter __rotate(_ForwardIter __first,
                      _ForwardIter __middle,
                      _ForwardIter __last,
                      _Distance*,
                      forward_iterator_tag)
{
    if (__first == __middle)
        return __last;
    if (__last  == __middle)
        return __first;

    _ForwardIter __first2 = __middle;
    do {
        swap(*__first++, *__first2++);  //
        if (__first == __middle)
            __middle = __first2;
    } while (__first2 != __last);

    _ForwardIter __new_middle = __first;
    __first2 = __middle;

    while (__first2 != __last)
    {
        swap (*__first++, *__first2++);  //
        if (__first == __middle)
            __middle = __first2;
        else if (__first2 == __last)
            __first2 = __middle;
    }

    return __new_middle;
}

//②利用ba=(br)^T(ar)^T=(arbr)^T，通过三次反转字符串。
//（即上述思路一，首先对序列前部分逆序，再对序列后部分逆序，再对整个序列全部逆序）
template <class _BidirectionalIter, class _Distance>
_BidirectionalIter __rotate(_BidirectionalIter __first,
                            _BidirectionalIter __middle,
                            _BidirectionalIter __last,
                            _Distance*,
                            bidirectional_iterator_tag)
{
    __STL_REQUIRES(_BidirectionalIter, _Mutable_BidirectionalIterator);
    if (__first == __middle)
        return __last;
    if (__last  == __middle)
        return __first;

    __reverse(__first,  __middle, bidirectional_iterator_tag());  //交换序列前半部分
    __reverse(__middle, __last,   bidirectional_iterator_tag());  //交换序列后半部分

    while (__first != __middle && __middle != __last)
        swap (*__first++, *--__last);   //整个序列全部交换

    if (__first == __middle)  //
    {
        __reverse(__middle, __last,   bidirectional_iterator_tag());
        return __last;
    }
    else {
        __reverse(__first,  __middle, bidirectional_iterator_tag());
        return __first;
    }
}
```

```
//④ 所有序号为 (i+t*k) % n (i为指定整数，t为任意整数)，
//会构成一个循环链（共有gcd(n,k)个，gcd为n、k的最大公约数），
//每个循环链上的元素只要移动一个位置即可，总共交换了n次。
template <class _RandomAccessIter, class _Distance, class _Tp>
_RandomAccessIter __rotate(_RandomAccessIter __first,
                           _RandomAccessIter __middle,
                           _RandomAccessIter __last,
                           _Distance *, _Tp *)
{
    __STL_REQUIRES(_RandomAccessIter, _Mutable_RandomAccessIterator);
    _Distance __n = __last   - __first;
    _Distance __k = __middle - __first;
    _Distance __l = __n - __k;
    _RandomAccessIter __result = __first + (__last - __middle);

    if (__k == 0)
        return __last;

    else if (__k == __l) {
        swap_ranges(__first, __middle, __middle);
        return __result;
    }

    _Distance __d = __gcd(__n, __k);    //令d为gcd(n,k)

    for (_Distance __i = 0; __i < __d; __i++) {
        _Tp __tmp = *__first;
        _RandomAccessIter __p = __first;

        if (__k < __l) {
            for (_Distance __j = 0; __j < __l/__d; __j++) {
                if (__p > __first + __l) {
                    *__p = *(__p - __l);
                    __p -= __l;
                }

                *__p = *(__p + __k);
                __p += __k;
            }
        }
        else {
            for (_Distance __j = 0; __j < __k/__d - 1; __j ++) {
                if (__p < __last - __k) {
                    *__p = *(__p + __k);
                    __p += __k;
                }

                *__p = * (__p - __l);
                __p -= __l;
            }
        }

        *__p = __tmp;
        ++__first;
    }

    return __result;
}
```

由于上述stl rotate源码中，方案④ 的代码，较复杂，难以阅读，下面是对上述第④ 方案的简单改写：

```
//对上述方案4的改写。
//④ 所有序号为 (i+t*k) % n (i为指定整数，t为任意整数)，....
//copyright@ hplonline && July 2011.04.18。
//July、sahala、yansha，updated，2011.06.02。
void my_rotate(char *begin, char *mid, char *end)
{
    int n = end - begin;
    int k = mid - begin;
    int d = gcd(n, k);
    int i, j;
    for (i = 0; i < d; i ++)
    {
        int tmp = begin[i];
        int last = i;

        //i+k为i右移k的位置，%n是当i+k>n时从左重新开始。
        for (j = (i + k) % n; j != i; j = (j + k) % n)     //多谢laocpp指正。
        {
            begin[last] = begin[j];
            last = j;
        }
        begin[last] = tmp;
    }
}
```

对上述程序的解释：关于第二个for循环中，j初始化为（i+）%n，程序注释中已经说了，i+k为i右移k的位置，%n是当i+k>n时从左重新开始。为什么要这么做呢?很简单，n个数的数组不管循环左移多少位，用上述程序的方法一共需要交换n次。当i+k>=n时i+k表示的位置在数组中不存在了，所以又从左边开始的(i+k)%n是下一个交换的位置。

1. 好比5个学生,，编号从0开始，即0 1 2 3 4，老师说报数，规则是从第一个学生开始，中间隔一个学生报数。报数的学生编号肯定是0 2 4 1 3。这里就相当于i为0，k为2，n为5

2. 然后老师又说，编号为0的学生出列，其他学生到在他前一个报数的学生位置上去，那么学生从0 1 2 3 4=》2 3 4 _ 1，最后老师说，编号0到剩余空位去，得到最终排位2 3 4 0 1。此时的结果，实际上就是相当于上述程序中左移k=2个位置了。而至于为什么让 编号为0 的学生出列。实际是这句：int last = i; 因为要达到这样的效果0 1 2 3 4 => 2 3 4 0 1，那么2 3 4 必须要移到前面去。怎么样，明白了么?。

关于本题，不少网友也给出了他们的意见，具体请参见此帖子**微软100题，维护地址**。

第五节、总结

如nossiac所说，对于这个数组循环移位的问题，真正最靠谱的其实只有俩种：一种是上文的思路一，前后部分逆置翻转法，第二种是思路三，即stl 里的rotate算法，其它的思路或方法，都是或多或少在向这俩种方法靠拢。

下期更新：程序员面试题狂想曲：第二章。时间：本周周日04.24晚。非常感谢各位朋友的，支持与关注。本人宣告：本程序员面试题狂想曲系列，永久更新。
本章完。

分享到：

上一篇：全新整理：微软、Google等公司的面试题及解答、第161-170题
下一篇：程序员编程艺术：第二章、字符串是否包含及匹配/查找/转换/拷贝问题

查看评论

223楼 programming_hard 6天前 17:27发表

各种崇拜佩服！ 仔细学习中！

222楼 wangjiejay 2013-02-12 17:23发表

最后面的那个STL有点不懂,求大神指点

221楼 t496036222 2013-01-20 10:59发表

无论是思想还是代码的严密性都对我有很大的启发，谢谢LZ的辛苦整理

220楼 lastdanc 2012-12-27 19:51发表

方法四：

对于正整数m、n，那么我们把它分成一个个互不影响的循环链，
当 j 固定所有序号为 (j + i * m) % n会构成一个循环链，
一共有gcd(n, m)个循环链，对每个循环链分别进行一次内循环就行了。
范围：
j=0:gcd(n, m)-1
i = 0：(n/gcd(n, m)) -1
例如：
其中gcd(n, m)为m、n的最大公约数

证明 i 的范围：
令：
n = K*gcd(n, m)
m = L*gcd(n, m)
K 和 L没有公约数

设经过 i 轮后重合则：
i*m = f*n
i = (f*n)/m = {f*K*gcd(n, m)}/{ L*gcd(n, m)} = {f*K} / L
K 和 L没有公约数，要使 i 最小，则 f = L
所以 i = K = n/gcd(n, m)

可证j的范围为：
j = n/i

219楼 lipeng08 2012-12-24 00:05发表

首先很感谢博主的热心奉献。