<http://m.blog.csdn.net/blog/lovekatherine/1549866>

# [[原]编程珠玑中的问题(1)——向量旋转](http://m.blog.csdn.net/blog/lovekatherine/1549866)

2007-4-2阅读1241 [评论1](http://m.blog.csdn.net/blog/lovekatherine/1549866#comment)

摘自《Programming Pearls》2rd 第2章  
  
问题：将一个具有n个元素的一维向量想左旋转k个位置。  
例如，假设n=8,k=3,那么向量abcdefgh旋转后得到向量defghabc。  
程序给定的限制是仅使用几十字节的微小内存，花费与n成比例的时间来完成旋转。  
  
我的思考：  
  
    该问题看起来可以用一个典型的递归思路来解决。  
  
    例如题目中所给的例子，可以按如下步骤完成：  
  
    (1),将(abc)与原向量的最后三个元素进行一一置换，得到记过(fgh)de(abc)，即(fghde)abc。（这里的括号只是为了标记以提示重点，无其他意义）。  
  
    (2).现在可以观察到abc已移动到最终位置，下面只需对剩余的5个元素组成的向量(fghde)向左旋转3个位置，来变为(defgh)，如此，原始问题转化为一个规模更小的子问题。  
  
    (3).递归的终止条件？不难看出，当剩余的元素数量为k的2倍时，执行完(1)中的置换后，就已完成了旋转，因此不需要进行递归。此外，当最初的k%n的结果为0时，己不需要进行置换，也不需要进行递归——原向量与旋转后的向量相同。  
  
    然而，上面的思考是不全面的：即始终假设每次递归中剩余元素的数量n'至少是k的2倍。事实上一定会出现n'<2k的情况的，这种情况下需要进行一些变形，才能继续递归。方法如下  
  
    由于n'<2k，令k'=n‘-k<k , 再以n',k'为参数完成上述的步骤(1);之后，此时，n'中的最后k个元素已移动至最终位置，因此需要对剩余的n'-k'个元素进行递归，其递归参数为n,k-k' 。  
  
上述算法的实现如下：

#include <stdio.h>  
#include <stdlib.h>  
  
typedef unsigned int size\_t;  
typedef char ELEMENT;  
  
int  Min( int A,int B);  
void SwapElement(char \* str,  int index1  , int index2);  
void Swap(char \* str,size\_t size, int cnt);  
void Whirl    (char \* str,  size\_t size , int cnt);  
  
int main (int argc,char \* argv[])  
{  
  
    ELEMENT target[]={'a','b','c','d','e','f','g'};  
  
  
    Whirl(target,sizeof(target),atoi(argv[1]) );  
  
    int index;  
  
    for(index=0; index < sizeof(target);index++)  
        printf("%c",target[index]);  
  
    printf(" ");  
  
}  
  
  
int Min( int A,int B)  
{  
      
    if(A<=B)  
        return A;  
    else  
        return B;  
}  
  
// swap the array 's two elements: str[index1]<=>str[index2]  
  
void SwapElement(ELEMENT \* str,int index1,int index2)  
{  
        char tmp=str[index1];  
        str[index1]=str[index2];  
        str[index2]=tmp;  
}  
  
  
// swap the array's two part:   str[0]~str[cnt-1] <=> str[size-cnt]~str[size-1]  
void Swap(ELEMENT \* str,size\_t size, int cnt)  
{      
    int index;  
    for(index=0;index<cnt;index++)  
        SwapElement(str,index,size-cnt+index);  
  
}  
  
//Whirl the array specified by str by number of cnt  
void Whirl    (ELEMENT \* str,  size\_t size , int cnt)  
{  
  
    cnt%=size;  
  
    if(cnt==0)  
        return;  
  
    if(cnt== size-cnt)  
    {      
        Swap(str,size,cnt);  
        return;  
    }  
    else if (cnt < size-cnt)  
    {      
        Swap(str,size,cnt);  
        Whirl(str,size-cnt,cnt);  
    }  
    else if(cnt > size-cnt)  
    {  
        int newcnt=size-cnt;  
        Swap(str,size,newcnt);  
        Whirl(str+newcnt,size-newcnt,cnt-newcnt);  
    }  
  
}

      
写完程序后，参看了书中给出的思路，我想到的这种递归方法是其中的一种，作者给出的评论为：“根据这个递归的算法可以到得到一个非常优雅的程序，但是该程序要求编码细腻，还需要深思熟虑，以确保程序具有足够的效率。  
  
此外，作者还给出了另外一个非常简洁而富有效率的算法，让我觉得不服不行，为啥我就做不到这么直接、清晰的思考问题呢？  
  
该方法的思路如下：  
  
    将向量的元素视为数组元素，那么问题可以转化为将最初形式为ab数组，转化为形式为ba的数组；其中a是原数组的前k个元素，而b是剩余元素。  
      
   解决该问题， 只需三个步骤：  
  
    reverse(0,k-1)    //将a部分进行180度的旋转，即将a中位置对称的元素互换，得到r(a)b  
    reverse(k,n)       //将b部分进行180度的旋转 ，得到r(a)r(b)  
    reverse(0,n-1)   //将整个数组进行180度的旋转，得到r(r(a)r(b)) =ba  
  
算法的实现如下：

#include <stdio.h>  
#include <stdlib.h>  
  
typedef unsigned int size\_t;  
typedef char ELEMENT;  
  
void Reverse(ELEMENT \* str,size\_t size);  
void Whirl( ELEMENT \* str, size\_t size, int cnt);  
void SwapElement(ELEMENT \* str,int index1,int index2);  
  
int main (int argc,char \* argv[])  
{  
  
    char target[]={'a','b','c','d','e','f','g'};  
    Whirl(target,sizeof(target),atoi(argv[1]) );  
  
    int index;  
  
    for(index=0; index < sizeof(target);index++)  
        printf("%c",target[index]);  
  
    printf(" ");  
}  
  
void SwapElement(ELEMENT \* str,int index1,int index2)  
{  
        char tmp=str[index1];  
        str[index1]=str[index2];  
        str[index2]=tmp;  
}  
  
  
void Reverse(ELEMENT \* str,size\_t size)  
{  
    int index;  
  
    for(index=0;index<size/2;index++)  
        SwapElement(str,index,size-index-1);  
  
}  
  
void Whirl( ELEMENT \* str, size\_t size, int cnt)  
{  
  
    cnt%=size;  
  
    Reverse(str,cnt);  
  
    Reverse(str+cnt,size-cnt);  
  
    Reverse(str,size);  
}

从算法的实现也可以看出第二种算法的优越性，不仅有更少的代码量，而且逻辑更清晰。

<http://384444165.iteye.com/blog/1749345>

最近太忙了买了编程珠玑之后就翻过2页，今天正好@**neiddy**问我看里面的题目，就花了点时间看看做一下。自己思考后跟书上讲的第二种有效方法很想，考虑后就在想执行效率，所以写了代码来看一下，最后需要做⌈length/rotatelength⌉ or +1（就不写那么复杂了）次此swap，每次swap做routateLength次交换，因此时间复杂度是O(n)，空间复杂度O(1)。其实第一种实现思路较为直观，开始思考了这样的替换，觉着复杂没有深入思考，其实也是很好的方法，时间复杂度低。看到第三种方法的时候就觉着自己弱爆了，好吧，说题目吧，我实现了第二种。怀着羡慕嫉妒恨写了第三种漂亮的算法。

说明：

因为每个循环交换需要用到一次temp，后两个算法每个循环只有2个值，但是杂技算法那这个每个循环可能有很多，因此减少了对temp的赋值次数，~~带来一定的性能提升（这里说的如果每个部分都是大数据的话~~ （纠正这个错误，实际的实现中肯定只会做地址的赋值，即便是大数据不可能做关于内存块的移动，因此这部分虽然减少了交换次数，但是由于下面提到的取余指令问题依然造成了性能的衰减，但是不会很明显，也不是在算法分析中需要太关注的环节，但是如果必须做内存块交换或外存空间的话需要谨慎选择算法，并尽量考虑使用其他的方法解决这个限制），如果只是交换char的话取余数并赋值为四条指令，远比加减和赋值多，因此性能反而得不到提升，这都属于细节了，如果真的是用大数据的话就要考虑牺牲一定的可读性了）。

题目：（原书11页问题B）

将一个n元一维向量向左旋转i个为孩子。例如，当n=8且i=3时，向量abcdefgh左旋转为defghabc。简单的代码使用一个n元的中间向量在n步完成该工作。你能否使用数十个额外字节的存储空间，在正比于n的时间内完成向量的旋转？

by me，第二种实现方法

**C代码**

1. /\*
2. \* vectorrotate.c
3. \*
4. \*  Created on: Dec 17, 2012
5. \*      Author: Jason\_wbw
6. \*/
8. #include <stdio.h>
9. #include <string.h>
11. /\*\*
12. \* 交换vector中的两部分：
13. \* 交换前的四部分[firstBegin, firstBegin+swapLength-1],
14. \*              [firstBegin+swapLength, secondBegin-1],
15. \*              [secondBegin, secondBegin+swapLength-1],
16. \*              [secondBegin+swapLength, end]
17. \* 交换内容为第一和第三部分，交换后如下
18. \*              [secondBegin, secondBegin+swapLength-1],
19. \*              [firstBegin+swapLength, secondBegin-1],
20. \*              [firstBegin, firstBegin+swapLength-1],
21. \*              [secondBegin+swapLength, end]
22. \*/
23. **void** swapVector(**char**\* vector, **int** firstBegin, **int** secondBegin, **int** swapLength){
24. **char** temp;
25. **int** i;
26. **for**(i=0; i<swapLength;i++){
27. temp = vector[firstBegin+i];
28. vector[firstBegin+i] = vector[secondBegin+i];
29. vector[secondBegin+i] = temp;
30. }
31. }
33. /\*\*
34. \* 向量的左旋转
35. \* 算法如下：
36. \* 按照position将vector分成ab两部分，最终需要变成ba。
37. \* [1]若a长度小于b，将b分为b1b2，其中b2长度等于a。交换a和b2变为b2b1a，再递归返回到判断修改b2b1为b1b2。
38. \* [2]若a长度大于b，将a分为a1a2，其中a1长度等于b。交换a2和b变为ba2a1，再递归返回到判断修改a2a1为a1a2。
39. \* [3]若a长度等于b，交换a、b，算法结束。
40. \*
41. \* 'vector'   要旋转的字符串数组
42. \* 'position' 旋转几个位置
43. \*
44. \* return 0 表示position越界；1 表示成功左旋转
45. \*/
46. **int** route(**char**\* vector, **int** position){
47. **if**(position<=0 || position>=strlen(vector)){
48. **return** 0;
49. }
51. **int** firstPosition = 0;                          //算法判断情况[1]中a的首地址
52. **int** headerLength = position;                    //算法判断情况[1]中a的长度
53. **int** enderLength = strlen(vector)-position;      //算法判断情况[1]中b的长度
55. **while**(headerLength!=enderLength){
56. **if**(headerLength<enderLength){            //算法判断情况[1]
57. swapVector(vector, firstPosition, firstPosition+enderLength, headerLength);
58. enderLength -= headerLength;
59. }**else**{                                   //算法判断情况[2]
60. swapVector(vector, firstPosition, firstPosition+headerLength, enderLength);
61. firstPosition = enderLength;
62. headerLength -= enderLength;
63. }
64. }
66. //算法判断情况[3]
67. swapVector(vector, firstPosition, firstPosition+headerLength, headerLength);
69. **return** 1;
70. }
72. /\*\*
73. \* test
74. \*/
75. **int** main(){
77. **char** vector[8] = {'a','b','c','d','e','f','g','h'};
78. printf("before rotate : %s\n",vector);
79. **int** result = route(vector,3);
80. **if**(result==1){
81. printf("after rotate : %s",vector);
82. }
83. **return** 0;
84. }

羡慕嫉妒恨的漂亮算法：

**Java代码**

1. /\*
2. \* beautiful\_rotate.c
3. \*
4. \*  Created on: Dec 18, 2012
5. \*      Author: Jason\_wbw
6. \*/
8. #include <stdio.h>
10. /\*\*
11. \* 将vector逆序，比如vector为abcde，逆序后为edcba
12. \*/
13. **void** reverse(**char**\* vector, **int** begin, **int** end){
14. **while**(begin < end){
15. **char** temp = vector[begin];
16. vector[begin] = vector[end];
17. vector[end] = temp;
18. begin++;
19. end--;
20. }
21. }
23. /\*\*
24. \* test
25. \* 测试左旋转位数为3，先对0-2进行逆序，再对3-7逆序，最后整体逆序完成了从ab序列变成ba序列
26. \*/
27. **int** main(){
29. **char** vector[8] = {'a','b','c','d','e','f','g','h'};
30. printf("before rotate : %s\n",vector);
31. reverse(vector, 0, 3-1);
32. reverse(vector, 3, 8-1);
33. reverse(vector, 0, 8-1);
34. printf("after rotate : %s",vector);
35. **return** 0;
36. }

by @**neiddy** 实现的第一种书上叫做杂技的算法，很好理解。这种算法每次循环只需要给temp（代码中的buffer做一次赋值，可以减少赋值的次数）

**C代码**

1. #include <stdio.h>
3. **void** string\_rotate( **char**\* str, **int** len, **int** n )
4. {
5. **int** i = 0;
6. **int** cnt = 0;
7. n %= len;
8. **for** ( i = 0; i < n && cnt < len; i++ )
9. {
10. **int** index = i;
11. **char** buf = str[i];
12. **while** (1)
13. {
14. **int** temp = ( index + n ) % len;
15. cnt++;
16. **if** ( temp == i )
17. {
18. str[index] = buf;
19. **break**;
20. }
21. **else**
22. {
23. str[index] = str[temp];
24. }
25. index = temp;
26. }
27. }
28. }
30. **int** main ( **void** )
31. {
32. **char** str[] = "12345678";
33. string\_rotate( str, 5, 2 );
34. printf("%s\n",str);
35. }

<http://my.oschina.net/renhc/blog/89298>

**问题描述**

请将一个具有n个元素的一维向量向左旋转i个位置。例如，假设n=8，i=3，那么向量abcdefgh旋转之后得到向量defghabc。简单编码使用一个具有n个元素的中间向量分n步即可完成此作业。你可以仅使用几十字节的微小内存，花费与n成比例的时间来旋转该向量吗？

**解决思路**

方案一：

将向量x中的前i个元素复制到一个临时数组中，接着将余下的n-i个元素左移i个位置，然后再将前i个元素从临时数组中复制回x中的后面位置。

该方案使用了i个额外的位置，如i足够大，过于浪费空间。

方案二：

定义一个函数来将x向左旋转一个位置，然后调用该函数i次。

该方案需要将数组移到i将，过于浪费时间。

方案三：

先将x[0]移临时变量t中，然后将x[i]移到x[0]中，x[2i]移到x[i]中，依次类推，直到我们又回到从x[0]中提取元素，不过在这时我们要从t中提取元素，然后结束该过程。当i=3，n=12时，该阶段将以下面的次序移到各个元素。

如果该过程不能移动所有的元素，那么我们再从x[1]开始移动，一直依次进行下去，直到移动了所有的元素时为止。

该方案过于精巧，像书中所说的一样堪称巧妙的杂技表演，非常容易出错。

方案四：

旋转向量x实际上就是将向量ab的两个部分交换为向量ba，这里a代表x的前i个元素。假设a比b短。将b分割成bl和br，使br的长度和a的长度一样。交换a和br，将ablbr转换成brbla。因为序列a已在它的最终位置了，所以我们可以集中精力交换b的两个部分了。由于这个新问题和原先的问题是一样的，所以我们以递归的方式进行解决。

该方案要求编码细腻，还需要深思熟虑，以确保程序具有足够的效率。

方案五：（最佳）

将这个问题看作是把数组ab转换成数组ba吧，但同时也假定我们具有在数组中转置指定部分元素的函数。我们先从ab开始，转置a得到arb，再转置b得到arbr，然后再转置整个arbr得到（arbr）r，实际上就是ba。

reverse(0, i-1)   /\*cbadefgh\*/  
reverse(i, n-1)  /\*cbahgfed\*/  
reverse(0, n-1) /\*defghabc\*/

该转置代码在时间和空间上都很有效，并且是这么简短和简单，想出错都很难。

书中还提供了将10个元素的数组向上旋转5个位置的手摇法例子：先是掌心对着你自己，左手在右手上面，如图所示

**代码实现**

[?](http://my.oschina.net/renhc/blog/89298)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47 | #include <stdio.h>    void swap(int \*p, int \*q);  void reverse(int \*vector, int index\_begin, int index\_end);  void revrot(int \*vector, int len, int step);    int main(int argc, char \*\*argv)  {     int  step = 3;     int  i = 0;     int  vector[1024] = {1,2,3,4,5,6,7,8};       revrot(vector, 8, step);       printf("after revolve: ");     for(i = 0; i < 8; i++)     {        printf("%d ", vector[i]);     }     printf("\n");  }    void swap(int \*p, int \*q)  {     int temp;       temp = \*p;     \*p = \*q;     \*q = temp;  }    void reverse(int \*vector, int index\_begin, int index\_end)  {     while(index\_begin < index\_end)     {        swap(vector + index\_begin, vector + index\_end);        index\_begin++;        index\_end--;     }  }    void revrot(int \*vector, int len, int step)  {     reverse(vector, 0, step - 1);     reverse(vector, step, len - 1);     reverse(vector, 0, len - 1);  } |