数组

七月算法 邹博

2015年4月14日

数组

- 仅指存储方式为数组，非考察语言级数组的问题
- 有些题目非常难，甚至是NP难解的
  - 后面将有NP的实例
- 往往有很难的面试压轴大题
  - 完美洗牌算法
- 本次目标：
  - 除了学会算法本身，在这个过程中，探讨算法如何设计？理清从特殊到一般的分析思路。

寻找和为定值的两个数

- 输入一个数组A[0…N-1]和一个数字Sum，在数组中查找两个数A_i,A_j，使得$A_i+A_j=Sum$。

暴力求解

- 从数组中任意选取两个数x,y，判定它们的和是否为输入的数字M。时间复杂度为$O(N^2)$，空间复杂度O(1)。

稍好一点的方法

- 如果数组是无序的，先排序(NlogN)，然后用两个指针i，j，各自指向数组的首尾两端，令i=0，j=n-1，然后i++，j--，逐次判断a[i]+a[j]是否等于Sum：
  - 如果a[i]+a[j]>sum，则i不变，j--；
  - 如果a[i]+a[j]<sum，则i++，j不变；
- 数组无序的时候，时间复杂度最终为O(NlogN+N)=O(NlogN)。

Code

```c
void TwoSum(int data[], unsigned int length, int sum)
{
    //sort(s, s+n);   如果数组非有序的，那就事先排好序O(N log N)
    int begin = 0;
    int end = length - 1;
    //两端扫描法，很经典的方法，O(N)
    while (begin < end)
    {
        long currSum = data[begin] + data[end];

        if (currSum == sum)
        {
            printf("%d %d\n", data[begin], data[end]);
            //如果需要所有满足条件的数组对，则需要加上下面两条语句：
            //begin++
            //end--
            break;
        }
        else{
            if (currSum < sum)
                begin++;
            else
                end--;
        }
    }
}
```

讨论：Hash方案的可行性

- 算法步骤
  - 选择适当的Hash函数，对原数组建立Hash结构
  - 遍历数组a[i]，计算Hash(Sum-a[i])是否存在
- 算法可行性
  - 时间复杂度，空间复杂度

和为定值的m个数

- 已知数组A[0…N-1]，给定某数值sum，找出数组中的若干个数，使得这些数的和为sum。
- 布尔向量x[0…N-1]
  - x[i]=0表示不取A[i]，x[i]=1表示取A[i]
  - 假定数组中的元素都大于0：A[i] > 0
  - 这是个NP问题！

分析方法

- 直接递归法（枚举）
- 分支限界
- 存在负数的处理办法

直接递归法

```
1: 1 2 3 4
2: 1 4 5
3: 2 3 5
```

```
int a[] = {1,2,3,4,5};
int size = sizeof(a) / sizeof(int);
int sum = 10;   //sum为计算的和

//x[]为最终解，i为考察第x[i]是否加入，has表示当前的和
void EnumNumber(bool* x, int i, int has)
{
    if(i >= size)
        return;
    if(has + a[i] == sum)
    {
        x[i] = true;
        Print(x);
        x[i] = false;
    }
    x[i] = true;
    EnumNumber(x, i+1, has+a[i]);
    x[i] = false;
    EnumNumber(x, i+1, has);
}

int _tmain(int argc, _TCHAR* argv[])
{
    bool* x = new bool[size];
    memset(x, 0, size);
    EnumNumber(x, 0, 0);
    delete[] x;
    return 0;
}
```

考虑对于分支如何限界

- 前提：数组A[0...N-1]的元素都大于0
- 考察向量x[0...N-1]，假定已经确定了前i个值，现在要判定第i+1个值x[i]为0还是1。
- 假定由x[0...i-1]确定的A[0...i-1]的和为has；
- A[i,i+1,...N-1]的和为residue（简记为r）；
  - has+a[i]≤sum并且has+r≥sum：x[i]可以为1；
  - has+(r-a[i])>= sum：x[i]可以为0；

分支限界法

1:	1	2	3	4	5	6	9	10	
2:	1	2	3	4	5	7	8	10	
3:	1	2	3	4	6	7	8	9	
4:	1	2	3	7	8	9	10		
5:	1	2	4	6	8	9	10		
6:	1	2	5	6	7	9	10		
7:	1	3	4	5	8	9	10		
8:	1	3	4	6	7	9	10		
9:	1	3	5	6	7	8	10		
10:	1	4	5	6	7	8	9		
11:	1	5	7	8	9	10			
12:	2	3	4	5	7	9	10		
13:	2	3	4	6	7	8	10		
14:	2	3	5	6	7	8	9		
15:	2	4	7	8	9	10			
16:	2	5	6	8	9	10			
17:	3	4	6	8	9	10			
18:	3	5	6	7	9	10			
19:	4	5	6	7	8	10			
20:	6	7	8	9	10				

```
int a[] = {1,2,3,4,5,6,7,8,9,10};
int size = sizeof(a) / sizeof(int);
int sum = 40;   //sum为计算的和

//x[]为最终解，i为考察第x[i]是否加入，has表示当前的和
//residue是剩余数的全部和
void FindNumber(bool* x, int i, int has, int residue)
{
    if(i >= size)
        return;
    if(has + a[i] == sum)
    {
        x[i] = true;
        Print(x);
        x[i] = false;
    }
    else if((has + residue >= sum) && (has + a[i] <= sum))
    {
        x[i] = true;
        FindNumber(x, i+1, has+a[i], residue-a[i]);
    }
    if(has + residue - a[i] >= sum)
    {
        x[i] = false;
        FindNumber(x, i+1, has, residue-a[i]);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    int residue = Sum(a, size);
    bool* x = new bool[size];
    memset(x, 0, size);
    FindNumber(x, 0, 0, residue);
    delete[] x;
    return 0;
}
```

数理逻辑的重要应用：分支限界的条件

- 分支限界的条件是充分条件吗？
- 在新题目中，如何发现分支限界的条件。
  - 学会该方法，比此问题本身更重要

考虑负数的情况

- 枚举法肯定能得到正确的解
- 如何对负数进行分支限界?
  - 可对整个数组A[0…N-1]正负排序，使得负数都在前面，正数都在后面，使用剩余正数的和作为分支限界的约束：
  - 如果A[i]为负数：如果全部正数都算上还不够，就不能选A[i]；
  - 如果递归进入了正数范围，按照数组是全正数的情况正常处理；
  - 注：正负排序马上要讲到

带负数的分支限界

A = {-3, -5, -2, 4, 2, 1, 3}
sum = 5

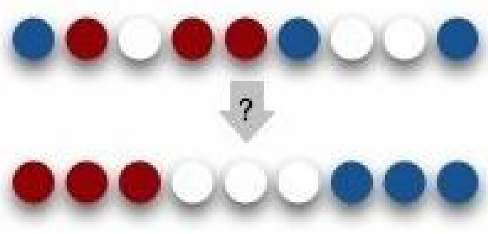
1:	-3	-2	4	2	1	3
2:	-3	4	1	3		
3:	-5	4	2	1	3	
4:	-2	4	2	1		
5:	-2	4	3			
6:	4	1				
7:	2	3				

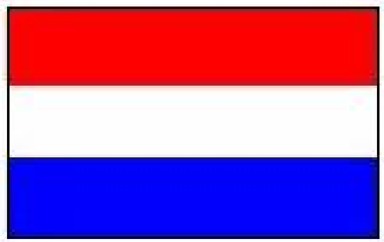
```
int _tmain(int argc, _TCHAR* argv[])
{
    int positive,negative;
    Sum(a, size,negative, positive);
    bool* x = new bool[size];
    memset(x, 0, size);
    FindNumber2(x, 0, 0, negative, positive);
    delete[] x;
    return 0;
}
```

```
//residue剩余的所有正数的和
void FindNumber2(bool* x, int i, int has, int negative, int positive)
{
    if(i >= size)
        return;
    if(has + a[i] == sum)
    {
        x[i] = true;
        Print(x);
        x[i] = false;
    }

    if(a[i] >= 0)
    {
        if((has + positive >= sum) && (has + a[i] <= sum))
        {
            x[i] = true;
            FindNumber2(x, i+1, has+a[i], negative, positive-a[i]);
            x[i] = false;
        }
        if(has + positive - a[i] >= sum)
        {
            x[i] = false;
            FindNumber2(x, i+1, has, negative, positive-a[i]);
        }
    }
    else
    {
        if(has + x[i] + positive >= sum)
        {
            x[i] = true;
            FindNumber2(x, i+1, has+a[i], negative-a[i], positive);
            x[i] = false;
        }
        if((has + negative <= sum) && (has + positive >= sum))
        {
            x[i] = false;
            FindNumber2(x, i+1, has, negative-a[i], positive);
        }
    }
}
```

荷兰国旗问题

- 现有红、白、蓝三个不同颜色的小球，乱序排列在一起，请重新排列这些小球，使得红白蓝三色的同颜色的球在一起。这个问题之所以叫荷兰国旗，是因为我们可以将红白蓝三色小球想象成条状物，有序排列后正好组成荷兰国旗。





问题分析

- 问题转换为：给定数组A[0…N-1]，元素只能取0、1、2三个值，设计算法，使得数组排列成“00…0011…1122…22”的形式。
- 借鉴快速排序中partition的过程。定义三个指针：begin=0、current=0、end=N-1；
- A[cur]==2，则A[cur] 与A[end]交换，end--，cur不变
- A[cur]==1，则cur++，begin不变，end不变
- A[cur]==0，则：
  - 若begin==cur，则begin++，cur++
  - 若begin≠cur，则A[cur]与A[begin]交换，begin++，cur不变

Code1

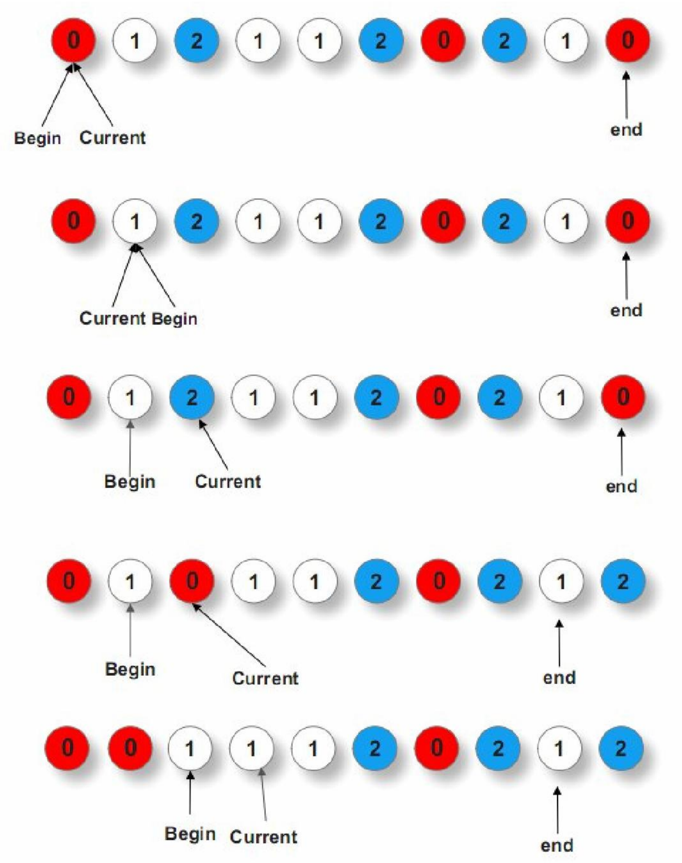
- A[cur]==2，则A[cur] 与A[end]交换，end--，cur不变
- A[cur]==1，则cur++，begin不变，end不变
- A[cur]==0，则：
  - 若begin==cur，则begin++，cur++
  - 若begin≠cur，则A[cur]与A[begin]交换，begin++，cur不变

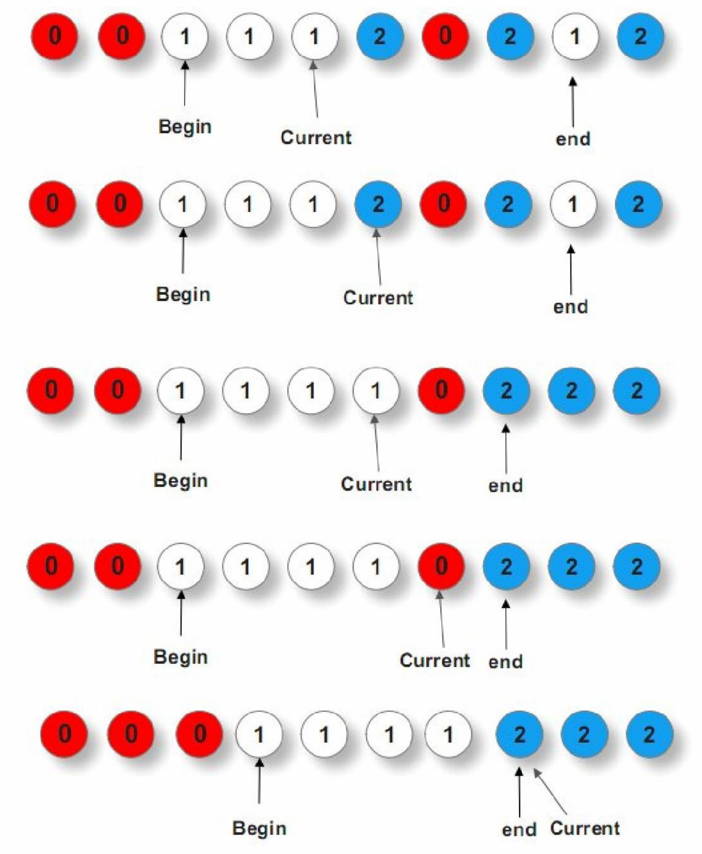
```
void Holland(int* a, int length)
{
    int begin = 0;
    int current = 0;
    int end = length - 1;
    while(current <= end)
    {
        if(a[current] == 2)
        {
            swap(a[end], a[current]);
            end--;
        }
        else if(a[current] == 1)
        {
            current++;
        }
        else// if(a[current == 0])
        {
            if(begin == current)
            {
                begin++;
                current++;
            }
            else
            {
                swap(a[current], a[begin]);
                begin++;
            }
        }
    }
}
```

进一步的考虑：略做优化

- current扫过的位置，即：[begin,cur)区间内，一定没有2
  - 在前面的A[cur]==2中，已经被替换到数组后面了
- 因此：A[begin]要么是0，要么是1，不可能是2
- 考察begin指向的元素的值：
- 若begin≠cur，则必有A[begin]=1
- 用归纳法，使用begin/cur/end的演示过程证明上述结论
- 因此，当A[cur]==0时，
  - 若begin≠cur，因为A[begin]==1，则交换后，A[cur]==1，此时，可以**cur++**；

优化后的图示演示


Begin Current
end
Current Begin
end
Begin Current
end
Begin Current
end
Begin Current
end


Begin Current
end
Begin Current
end
Begin Current
end
Begin Current end
Begin end Current

Code2

- A[cur]==2，则A[cur]与A[end]交换，end--，cur不变
- A[cur]==1，则cur++，begin不变，end不变
- A[cur]==0，则：
  - 若begin==cur，则begin++，cur++
  - 若begin≠cur，则A[cur]与A[begin]交换，begin++，cur++

```
void Holland(int* a, int length)
{
    int begin = 0;
    int current = 0;
    int end = length - 1;
    while(current <= end)
    {
        if(a[current] == 2)
        {
            swap(a[end], a[current]);
            end--;
        }
        else if(a[current] == 1)
        {
            current++;
        }
        else// if(a[current == 0])
        {
            if(begin == current)
            {
                begin++;
                current++;
            }
            else
            {
                swap(a[current], a[begin]);
                begin++;
                current++;
            }
        }
    }
}
```

Code2'

```
void Holland(int* a, int length)
{
    int begin = 0;
    int current = 0;
    int end = length - 1;
    while(current <= end)
    {
        if(a[current] == 2)
        {
            swap(a[end], a[current]);
            end--;
        }
        else if(a[current] == 1)
        {
            current++;
        }
        else// if(a[current == 0])
        {
            //1、或者用更直接的判断if(a[current] != a[begin]);
            //2、因为不等的次数远远大于相等的次数，可以直接删去该判断
            if(current != begin)
                swap(a[current], a[begin]);
            begin++;
            current++;
        }
    }
}
```

荷兰国旗问题带来的思考

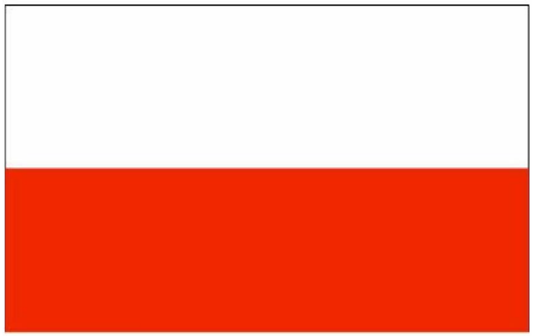
- 在begin/cur/end的循环中，cur遇到0和遇到2，begin和end的处理方式不对称
  - 遇到0：begin++,cur++
  - 遇到2：end--，cur不变
- 若初值给定如下：
  - begin=0、current=N-1、end=N-1，如何完成代码？

Code3

```
void Holland(int* a, int length)
{
    int begin = 0;
    int end = length - 1;
    int current = end;
    while(current >= begin)
    {
        if(a[current] == 2)
        {
            swap(a[end], a[current]);
            end--;
            current--;
        }
        else if(a[current] == 1)
        {
            current--;
        }
        else// if(a[current == 0])
        {
            swap(a[current], a[begin]);
            begin++;
        }
    }
}
```

“乌克兰国旗”问题

- 如果是分成两部分呢?
  - 给定整数数组，要求奇数在前，偶数在后
    - 奇偶排序
  - 给定实数数组，要求负数在前，正数在后
    - 正负排序

讨论：荷兰国旗问题的其他方案

- 荷兰国旗中，0，1，2分别计数，然后根据三个计数值，赋值数组；
  - 是否可行
- 将（0,1）（2）根据快速排序的Patition，分成两堆（不妨使用1.5或者其他数作为PivotKey），然后，将（0）（1）根据快速排序的Patition，分成两堆（不妨使用0.5或者其他数作为PivotKey）。

完美洗牌算法

- 长度为2n的数组{a1,a2,a3,...,an,b1,b2,b3,...,bn}，经过整理后变成{a1,b1,a2,b2,....,an,bn}，要求时间复杂度O(n)，空间复杂度O(1)。

步步前移

- 观察变换前后两个序列的特点，我们可做如下一系列操作：
- 第①步确定b1的位置，即让b1跟它前面的a2，a3，a4交换：
  - a1，b1，a2，a3，a4，b2，b3，b4
- 第②步、接着确定b2的位置，即让b2跟它前面的a3，a4交换：
  - a1，b1，a2，b2，a3，a4，b3，b4
- 第③步、b3跟它前面的a4交换位置：
  - a1，b1，a2，b2，a3，b3，a4，b4
- b4已在最后的位置，不需要再交换。如此，经过上述3个步骤后，得到我们最后想要的序列。
- 移动n-1次，第i次将n-i个元素后移。时间复杂度为O（N^2）。

中间交换

- 每次让序列中最中间的元素进行交换。
- 对于a1，a2，a3，a4，b1，b2，b3，b4
- 第①步：交换最中间的两个元素a4，b1，序列变成：
  - a1，a2，a3，b1，a4，b2，b3，b4
- 第②步，让最中间的两对元素各自交换：
  - a1，a2，b1，a3，b2，a4，b3，b4
- 第③步，交换最中间的三对元素，序列变成：
  - a1，b1，a2，b2，a3，b3，a4，b4

中间交换

1 2 3 4 5 6 7 8 9 X

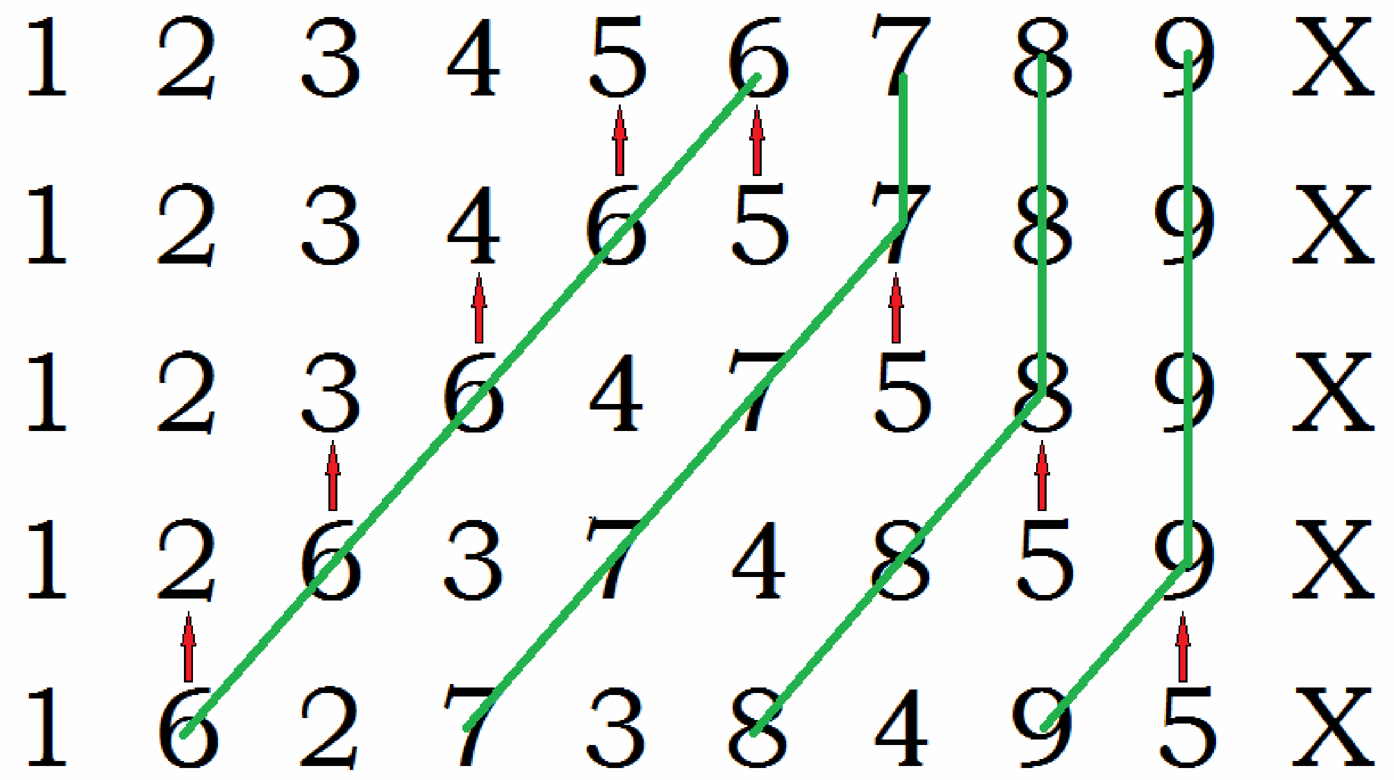
1 2 3 4 6 5 7 8 9 X

1 2 3 6 4 7 5 8 9 X

1 2 6 3 7 4 8 5 9 X

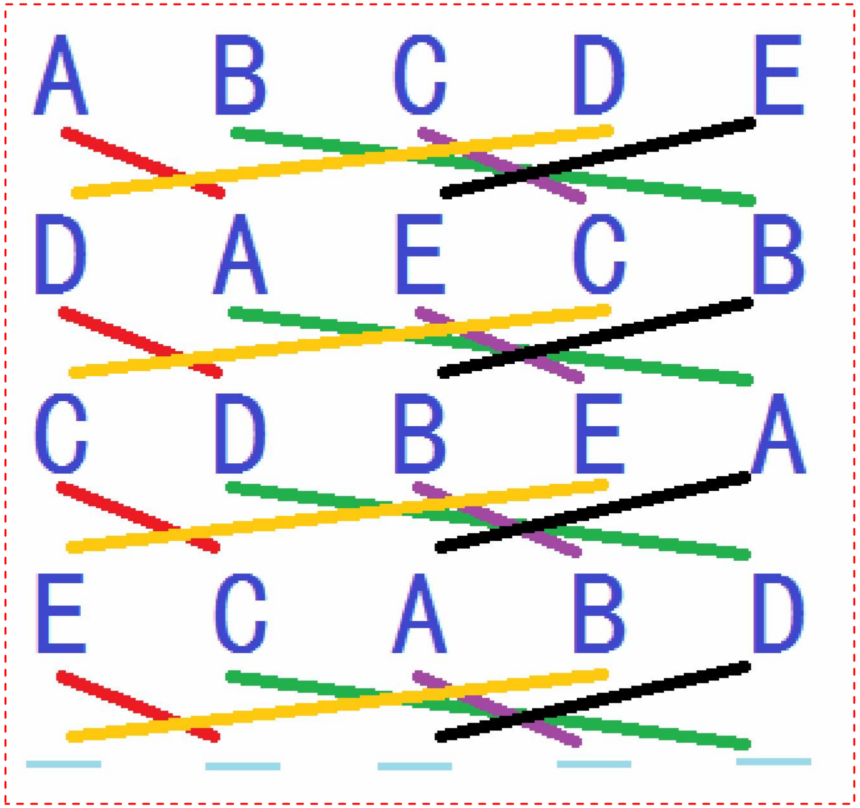
1 6 2 7 3 8 4 9 5 X

中间交换


1 2 3 4 5 6 7 8 9 X
1 2 3 4 6 5 7 8 9 X
1 2 3 6 4 7 5 8 9 X
1 2 6 3 7 4 8 5 9 X
1 6 2 7 3 8 4 9 5 X

玩乐中带来的算法思维

哈佛大学智商测试
请补填上第4行字母。
A B C D E
D A E C B
C D B E A
A
B
C
D
E



A B C D E
D A E C B
C D B E A
E C A B D

完美洗牌算法

- 2004年， microsoft的Peiyush Jain在他发表一篇名为：“A Simple In-Place Algorithm for In-Shuffle”的论文中提出了完美洗牌算法。

位置变换

- a1,a2,a3,...an,b1,b2,b3..bn → b1,a1,b2,a2,b3,a3...bn,an
- 设定数组的下标范围是[1..2n]。考察元素的最终位置：
- 以n=4为例，前n个元素中，
  - 第1个元素a1到了原第2个元素a2的位置，即1->2；
  - 第2个元素a2到了原第4个元素a4的位置，即2->4；
  - 第3个元素a3到了原第6个元素b2的位置，即3->6；
  - 第4个元素a4到了原第8个元素b4的位置，即4->8；
- 前n个元素中，第i个元素的最终位置为（2 * i）。
- 后n个元素，可以看出：
  - 第5个元素b1到了原第1个元素a1的位置，即5->1；
  - 第6个元素b2到了原第3个元素a3的位置，即6->3；
  - 第7个元素b3到了原第5个元素b1的位置，即7->5；
  - 第8个元素b4到了原第7个元素b3的位置，即8->7；
- 后n个元素，第i个元素的最终位置为：(2*(i - n)) - 1 = 2*i - 2*n-1 = (2*i) % (2*n+1)

两个圈

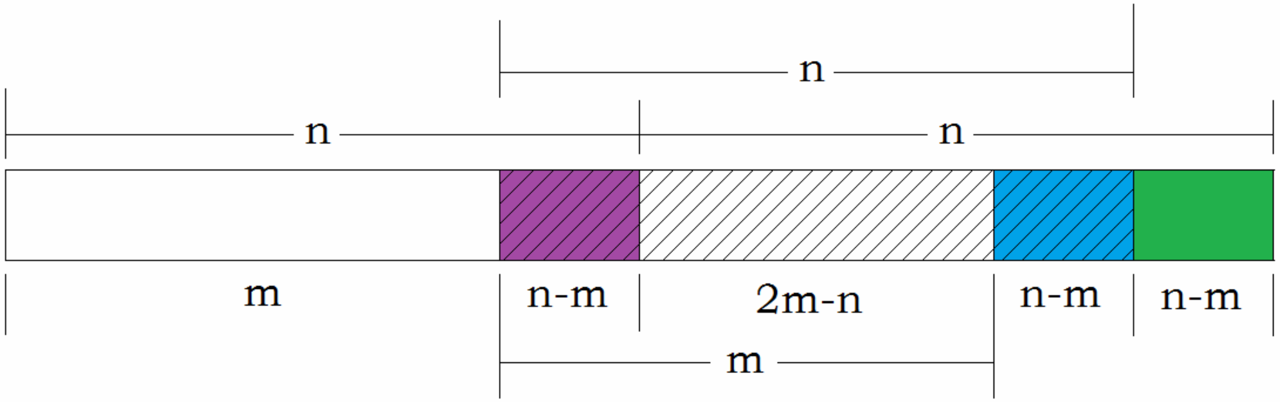
- 我们得到两个圈
- 1 → 2 → 4 → 8 → 7 → 5 → 1
- 3 → 6 → 3

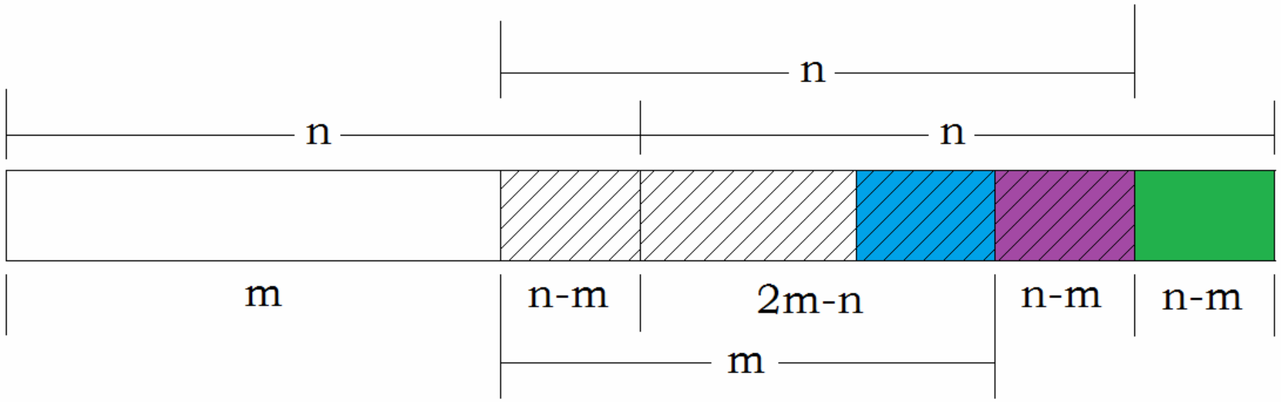
```
//数组下标从1开始，from是圈的头部，mod为 2 * n + 1
void CycleLeader(int *a, int from, int mod)
{
    int t,i;

    for(i = from * 2 % mod; i != from; i = i * 2 % mod)
    {
        t = a[i];
        a[i] = a[from];
        a[from] = t;
    }
}
```

K个圈

- 对于2*n =（3^k-1）这种长度的数组，恰好只有k个圈，且每个圈的起始位置分别是1,3,9，...3^(k-1)

若：2m可以写成3^k-1的形式





任意长度数组的完美洗牌算法

Input: An array $A[1, \ldots, 2n]$

Step 1. Find a $2m = 3^k - 1$ such that $3^k \leq 2n < 3^{k+1}$

Step 2. Do a right cyclic shift of $A[m+1, \ldots, n+m]$ by a distance m

Step 3. For each $i \in \{0, 1, \ldots, k-1\}$, starting at 3^i, do the cycle leader algorithm for the in-shuffle permutation of order $2m$

Step 4. Recursively do the in-shuffle algorithm on $A[2m+1, \ldots, 2n]$.

循环移位

- (AB)'=B'A'

```
//翻转字符串时间复杂度O(to - from)
void reverse(int *a, int from, int to)
{
    int t;
    for (; from < to; ++from, --to)
    {
        t = a[from];
        a[from] = a[to];
        a[to] = t;
    }
}
//循环右移num位 时间复杂度O(n)
void RightRotate(int *a, int num, int n)
{
    reverse(a, 1, n - num);
    reverse(a, n - num + 1, n);
    reverse(a, 1, n);
}
```

完美洗牌算法流程

- 输入数组A[1..2 * n]
- step 1 找到 2*m=3^k-1，且3^k≤2*n<3^(k+1)
- step 2 把a[m+1…m+n]那部分循环右移m位
- step 3 对每个i = 0,1,2..k - 1，3^i是每个圈的起始位置，做cycle_leader算法；
  - 注：因为子数组长度为m，所以对2*m+1取模
- step 4 对数组的剩余部分A[2*m+1.. 2*n]继续使用本算法。

完美洗牌代码

```
void PerfectShuffle2(int *a, int n)
{
    int n2, m, i, k, t;
    for (; n > 1;)
    {
        // step 1
        n2 = n * 2;
        for (k = 0, m = 1; n2 / m >= 3; ++k, m *= 3)
          ;
        m /= 2;
        // 2m = 3^k - 1 , 3^k <= 2n < 3^(k + 1)

        // step 2
        right_rotate(a + m, m, n);

        // step 3
        for (i = 0, t = 1; i < k; ++i, t *= 3)
        {
          cycle_leader(a , t, m * 2 + 1);
        }

        //step 4
        a += m * 2;
        n -= m;

    }
    // n = 1
    t = a[1];
    a[1] = a[2];
    a[2] = t;
}
```

依据

- 2是3的原根，2是9的原根
- $\{2^0,2^1\}=\{1,2\}$
- $\{2^0,2^1,2^2,2^3,2^4,2^5,2^6,2^7,2^8\} \bmod 9$
- $=\{1,2,4,8,7,5\}$
- 而 $\phi(9)=6$

附：算法原文

We show that, when $2n$ is of the form 3^k-1, we can easily determine the cycles of the in-shuffle permutation of order $2n$. We will need the following theorem from number theory:

Theorem 1 *If p is an odd prime and g is a primitive root of p^2, then g is a primitive root of p^k for any $k \geq 1$.*

A proof of this theorem can be found in [Nar00, p 20-21].

It can be easily seen that 2 is a primitive root of 9. From the above theorem it follows that 2 is also a primitive root of 3^k for any $k \geq 1$. This implies that the group $(Z/3^k)^*$ is cyclic with 2 being its generator.

Now let us analyse the cycles of an in-shuffle permutation when $2n = 3^k - 1$.

The cycle containing 1 is nothing but the group $(Z/3^k)^*$, which consists of all numbers relatively prime to 3^k and less than it.

Let $1 \leq s < k$. Consider the cycle containing 3^s. Every number in this cycle is of the form $3^s 2^t$ (*modulo* 3^k) for $1 \leq t \leq \varphi(3^k)$ (where φ is the Euler-totient function). Since 2 is a generator of $(Z/3^k)^*$, this cycle contains *exactly* the numbers less than 3^k which are divisible by 3^s but not by any higher power of 3.

This means that in an in-shuffle permutation of order $3^k - 1$, we have exactly k cycles with $1, 3, 3^2, \ldots, 3^{k-1}$ each belonging to a different cycle. Thus for these permutations, it becomes easy to pick the 'next' cycle in order to apply the cycle leader algorithm. Note that the length of the cycle containing 3^s is $\varphi(3^k)/3^s$, which helps us implement the cycle leader algorithm more efficiently.

进一步的思考

- 要求输出是a1,b1,a2,b2……an,bn，而完美洗牌算法输出是b1,a1,b2,a2,……bn,an,怎么办？
  - 先把a部分和b部分交换，或者最后再交换相邻的两个位置——不够美观。
  - 原数组第一个和最后一个不变，中间的2*(n-1)项用原始的完美洗牌算法。
- 逆完美洗牌问题：给定b1,a1,b2,a2,……bn,an，要求输出a1,a2,a3,……an,b1,b2,b3,……bn。
  - 既然完美洗牌问题可以通过若干圈来解决，那么，逆完美洗牌问题仍然存在是若干圈，并且2*n=（3^k-1）这种长度的数组恰好只有k个圈的结论仍然成立。
- 完美洗多付牌：给定a1,a2,……an, b1,b2,……bn, c1,c2,……cn，要求输出是c1,b1,a1,c2,b2,a2,……cn,bn,an
  - 2付牌的结论：2是群(Z/3^k)*最小生成元，且(3^k-1)这种长度的数组，恰好只有k个圈
  - 考察是否存在某数字p（如5、7、11、13等），使得数字3是群(Z/p^k)*的最小生成元，再验证p是否存在结论(p^k-1)这种长度的数组，恰好只有k个圈。
  - 提示：3是7的原根，是49的原根，于是3是7^k的原根

参考文献

- http://blog.csdn.net/v_JULY_v/article/details/6419466(和为定值的N个数)
- http://blog.csdn.net/jinyongqing/article/details/12054495 (和为定值的N个数)
- https://github.com/julycoding/The-Art-Of-Programming-By-July/blob/master/ebook/zh/02.03.md (和为定值的N个数)
- Peiyush Jain, A Simple In-Place Algorithm for In-Shuffle, Microsoft, July, 2004(完美洗牌)
- https://github.com/julycoding/The-Art-Of-Programming-By-July/blob/master/ebook/zh/02.09.md(完美洗牌)
- http://blog.csdn.net/caopengcs/article/details/10521603(完美洗牌)
- http://zh.wikipedia.org/wiki/%E6%AC%A7%E6%8B%89%E5%87%BD%E6%95%B0(欧拉函数)
- http://www.cnblogs.com/frog112111/archive/2012/08/13/2636334.html (欧拉函数，原根)
- http://blog.csdn.net/v_july_v/article/details/18824517(荷兰国旗问题)
- https://github.com/julycoding/The-Art-Of-Programming-By-July/blob/master/ebook/zh/02.07.md(荷兰国旗问题)
- https://github.com/julycoding/The-Art-Of-Programming-By-July/blob/master/ebook/zh/02.06.md(“乌克兰国旗”问题：奇偶排序)

我们在这里

- 更多算法面试题 7 七月算法 官网
  - http://www.julyedu.com/
    - 免费视频
    - 直播课程
    - 问答社区
- contact us：微博
  - @七月问答
  - @七月算法

感谢大家

恳请大家批评指正！