运用化归思想解决信息学中的数列问题

余林韵

指导教师

陈颖

福建省福州第一中学

高三 9 班

2007 年 11 月

摘　　要

数列是数学学科一个重要的分支，有着广泛的应用。从古至今有许多研究数列问题的经典论著。本文试图应用数学知识，从计算机解决问题的角度，结合信息学竞赛中的数列问题，展开分析、研究和讨论，提出解决数列问题的思维方法和技巧。向读者传达这样一个信息：解决数列问题不能墨守陈规，要敢于创新，联系所学知识，综合运用各种方法。

本文共分为五章：

第一章　基础数学

介绍解决数列问题涉及到的数学方法，为后面几章数列问题的解决奠定基础。

第二章　运用数列的“周期性”化归

本章选取了两道有代表性的例题，它们的背后隐藏着美妙的“周期性”。合理运用数列的“周期性”并结合一、二章提到的方法展开分析，大大降低了时间复杂度，有助于高效解决问题。这种方法对我们日后的解题也有所启发。

第三章　多元化方法化归解决数列问题

本章选取了 2007 年 NOI day 2 的第二题 Count 以及 Ural 上的 1396 号问题作为例题，在第三章合理运用数列“周期性”的基础上，介绍了通过化归、解决与数列相关的信息学问题的其他方法，主要有深入分析问题本质、寻找数列通项公式，分阶段考虑问题等。

第四章　总结

本章简述了数列问题在信息学竞赛中的重要地位，对之前四章的内容进行了分析、总结，并重复强调了本文中的重要思想方法。

关键词： 数列 化归 辗转相除法 快速幂运算 矩阵乘法 周期性 多元化方法

目　　录

前言

随着计算机逐渐步入人们的生活，网络日趋普及，“信息革命”的到来。信息学作为一门新兴的学科, 越来越引起世界各国的重视。跨入新世纪, 随着素质教育的全面推进、课程改革的全面实施，我国的信息学教育课程已从小学开始成为学生的必修科目，并一直延伸到大学。

信息学的学习需要掌握好数学物理等基础学科的知识，而通过信息学的学习和实践、运用，可培养严谨的逻辑思维能力，全面促进人的创新精神的培养和全面素质的提高。在信息学的学习中，我们经常会遇到许多数列问题，在人类文明的发展进程中，人们对数列的研究可谓历史悠久。人类在古代随着自然数、分数的概念和四则运算的产生，为了生产和生活的需要，开始了对数列的研究。在世界数学史上，中国、巴比伦、古希腊、埃及和印度等，都曾经研究过级数，中国古代数学名著《周髀算经》《九章算术》《孔子算经》《张邱建算经》等，对等差级数 $a+(a+b)+(a+2b)+(a+3b)+\ldots+[a+(n-1)b]$ 和等比级数 $a+a*q+a*q^2+a*q^3+\ldots+a*q^{n-1}$ 都列举出计算的例子，说明中国古代对级数的研究曾作出过一定的贡献。古老的《易经》一书中写道:“是故《易》有太极，是生两仪；两仪生四象，四象生八卦”，实际上，这种分割，已经寓有数学中等比数列的思想。正由于数列知识在解决生产生活实际问题中具有重要作用，因此，世界各国从来就没有停止过对它的研究。我国更是把数列作为学习高等数学的基础和培养学生能力的良好题材，列入了高中数学的必修内容。

参加过小学数学联赛的同学一定做过“找规律填数”这类趣题。而在信息学联赛中，也不乏很多有趣的数列题以及与数列相关的题目。本文将从中选取几道有代表性的题目，结合各题特性解题，从中找出一般规律，希望能起到抛砖引玉的作用。

解决数列问题，一方面需要解题者有着丰富的数学知识，另一方面需要解题者能够有着开拓性的思维，充分运用化归思想，大胆猜想，不断挖掘问题的本质。

化归，就是转化和归结，在解决问题时，人们常常将待解决的问题甲，通过某种转化过程，归结为一个已经解决或者比较容易解决的问题乙，然后通过乙

问题的解答返回去求得原问题甲的解答，这就是化归方法的基本思想。它的实质就是将新问题转化为已掌握的旧知识，然后进一步理解并解决新问题。

化归方法的要素有：化归对象，即对什么东西进行化归；化归目标，即化归到何处去；化归途径，即如何进行化归。

第一章　预备知识

1.1　矩阵乘法

有两个矩阵 A, B，$A = (a_{ij})_{n*m}, B = (b_{ij})_{m*t}$ 进行矩阵乘法 $A \cdot B$：

$$C = A \cdot B = (c_{ij})_{n*t}, c_{ij} = \sum_{k=1..m} a_{ik} * b_{kj}$$

观察公式可知矩阵乘法的时间复杂度为 $O(NMT)$，若矩阵的大小均为 N^2 的，那么复杂度就是 $O(N^3)$。当然，矩阵乘法还有更好的理论实现方法 {时间复杂度低于 $O(N^3)$，但是由于常数巨大，因此实现效果并不比 $O(N^3)$ 的算法优}。

矩阵乘法不满足交换律，但满足于结合律，因此，若有矩阵 $A = (a_{ij})_{n*n}$，计算 A^M 可以利用倍增思想进行计算：[1]

1.2　辗转相除法

1.2.1　辗转相除法

两数的最大公约数, 是指两数共有的约数之中最大的一个。a,b 的最大公约数记做 $\gcd(a, b)$ 。辗转相除法是利用以下性质来确定两个正整数 a 和 b 的最大公约数的：

a)　a 和其倍数之最大公约数为 a 。

b)　如果 q 和 r 是 a 除以 b 的商及余数, 即 $a = bq + r$ 。则 $\gcd(a, b) = \gcd(b, r)$

[1] 单位矩阵 E: 主对角元全为 1 ，非主对角元全为 0 , 即

$$E = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Algorithm 1 calculate A^M

```
1: function MatrixMulty(A,M)
2:     MatrixResult ← E
3:     MatrixX ← A
4:     Y ← M
5:     while Y > 0 do
6:         if odd number Y then
7:             MatrixResult ← MatrixResult · MatrixX
8:         end if
9:         MatrixX ← MatrixX · MatrixX
10:        Y ← Y div 2
11:    end while
12: end function
```

证明. a) 根据最大公约数的定义易知。

b) 设 $x = gcd(a, b), y = gcd(b, r)$

则有 $x|a$ 及 $x|b$，因此 $x|(a - bq)$ [2]，即 $x|r$ 及 $x|b$，故 $x|y$

而 $y|b$ 及 $y|r$，因此 $y|(bq + r)$，即 $y|a$ 及 $y|b$，故 $y|x$

因为 $x|y$ 且 $y|x$，所以 $x = y$，即 $\gcd(a, b) = \gcd(b, r)$ 。

□

Algorithm 2 Euclid's algorithm

```
1: function GCD(a,b)                          ▷ The g.c.d of a and b
2:     if b = 0 then
3:         return a
4:     else
5:         return GCD(b, a mod b)
6:     end if
7: end function
```

[2]同余的性质，下同

1.2.2　辗转相除法的时间复杂度分析

因为 $r < a/2$ [3]，故最多进行 $O(log_2 a)$ 次 gcd(x,y) 操作，故时间复杂度为 $O(log_2 a)$ 。

1.2.3　辗转相除法的扩展

已知 a,b，求一组 (x,y) 使得 $ax + by = \gcd(a, b)$ 。

记 q 和 r 是 a 除以 b 的商及余数, 即 $a = bq + r$ 。

如果我们已知一组 x,y 使得 $by + xr = \gcd(b, r)$，那么：

$$by + x(a - qb) = b(y - qx) + ax = \gcd(b, r) = \gcd(a, b)$$

故 $(x, y - qx)$ 是满足 $ax + by = \gcd(a, b)$ 的一组解。

Algorithm 3 Extended Euclid's algorithm

```
1:  function ExtendedGCD(a,b,x,y)
2:      if b = 0 then
3:          Begin
4:              x ← 1
5:              y ← 0
6:              Result ← a
7:          End
8:      else
9:          Begin
10:             Result ← ExtendedGCD(b, a mod b, y, x)
11:             y ← y − a div b ∗ x
12:         End
13:     end if
14: end function
```

[3] 若 $b \leqslant a/2$，则 $r < b \leqslant a/2$，若 $b > a/2$，则 $r = a - b < a - a/2 = a/2$

1.3　快速计算常系数线性递推数列的第 N 项

1.3.1　快速计算等比数列的第 N 项

由等比数列的通项公式 $a_n = a_1 q^{n-1}$ 可知，要快速求得 a_n，我们只需快速计算出 q^{n-1}。计算 q^{n-1} 可以使用 **快速幂运算**，把 $n-1$ 二进制展开，通过计算 $q^1, q^2, q^4, \ldots, q^{2^k}$ 可得。

Algorithm 4 快速幂运算 Calculate q to an n power

```
1:  function POWER(q,n)
2:      Result ← 0
3:      X ← q
4:      Y ← n
5:      while Y > 0 do
6:          if number Y then
7:              Result ← Result * X
8:          end if
9:          X ← X * X
10:         Y ← Y div 2
11:     end while
12: end function
```

这样，我们就在 O($log_2 N$) 的时间内快速计算出了等比数列的第 N 项。

1.3.2　快速计算广义 Fibonacci 数列的第 N 项

观察数列的前几项：

$$F_3 = F_1 + F_2 = a + b$$

$$F_4 = F_2 + F_3 = F_2 + (a + b) = a + 2b$$

$$F_5 = F_3 + F_4 = (a + b) + (a + 2b) = 2a + 3b$$

$$\ldots$$

也就是说，广义 Fibonacci 数列的任意一项都是由若干个 a 和若干个 b 加和组成的。

能否使用矩阵乘法来解决呢？

尝试用矩阵来表示数列的递推公式：由于要知道新的一项都需要之前的两项，因此我们把相邻两项组成一个矩阵，再利用矩阵间的关系得出：

$$\begin{pmatrix} F_{i-1} \\ F_i \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} F_{i-2} \\ F_{i-1} \end{pmatrix}$$

于是：

$$\begin{pmatrix} F_{n-1} \\ F_n \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} F_{n-2} \\ F_{n-1} \end{pmatrix}$$

$$= \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} F_{n-3} \\ F_{n-2} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^2 \cdot \begin{pmatrix} F_{n-3} \\ F_{n-2} \end{pmatrix}$$

$$= \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^2 \cdot \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} F_{n-4} \\ F_{n-3} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^3 \cdot \begin{pmatrix} F_{n-4} \\ F_{n-3} \end{pmatrix}$$

$$\cdots$$

$$= \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^{n-1} \cdot \begin{pmatrix} F_1 \\ F_2 \end{pmatrix}$$

这样我们就可以计算出 F_n。由于 $\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^{n-1}$ 可以在 O($log_2 N$) 的时间内求出，因此我们就顺利地在 O($log_2 N$) 的时间内求出广义 Fibonacci 数列的第 N 项。

1.3.3　快速计算常系数线性递推数列的第 N 项

通过对求解广义 Fibonacci 数列的第 N 项的过程我们可以发现，对于更加一般的常系数线性递推数列，我们也可以用类似的办法来解决问题：

如果已知数列 $\{a_n\}$ 的前 k 项，且任一项 a_n 与它的前 k 项间的关系，那么就可以建立一个 $k * 1$ 的矩阵来表示，并建立 $k * k$ 的矩阵来维系相邻矩阵间的关系，将 $k * k$ 的矩阵的 $n - k$ 次方乘上第一个矩阵即可求得数列的第 n 项，具体的，如果数列的递推公式为：

$$a_{n+k} = c_1 a_{n+k-1} + c_2 a_{n+k-2} + \ldots + c_k a_n$$

则 $k*k$ 的矩阵形态如下所示：

$$\begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & 0 & 1 \\ c_1 & c_2 & c_3 & \cdots & c_k \end{pmatrix}$$

该算法的时间复杂度为：$O(log_2(n-k)) * O(\text{ 矩阵乘法 }) = O(k^3 log_2 n)$ 。

第二章　合理运用数列的“周期性”

除常系数线性递推数列问题经常出现外，一些特殊的非线性递推数列问题在信息学竞赛中也屡见不鲜。对一般的非线性递推方程确定的数列没有通用的方法，涉及的问题也不仅仅是求通项（有时难以用显式表达出通项），往往是确定通项的某些性质。

然而，每个数列都有它的特性，这些特性往往是周期性或者存在阶段性的，对我们的解题很有帮助。本章通过两道例题，阐述如何通过深入的问题分析，合理运用“周期性”这一特点，成功化归，圆满解决问题。

2.1　例一 Dispute

问题描述：

数列 F_n 满足：

$F_0 = 0$

$F_n = g_{n,F_{n+1}}$

其中：$g_{x,y} = ((y-1)x^5 + x^3 - xy + 3x + 7y) \mod 9973$

给定 n，求解 F_n。[1]

数据规模：

$0 \leqslant n \leqslant 100000000$

分析：

由于 n 值最大可到达 10^8，因此单纯的递推显然难以承受。观察数列递推公式，发现它也不是线性的，不能简单地使用矩阵乘法。而细看数列的前几项：$3, 93, 2818, 968, 2132, \ldots$，似乎也找不出什么规律。

[1] Source:Ural 1309，为了方便运算，这个数列存在第 0 项

算法一：

在每个地方的分析都碰壁的情况下，使用暴力解决不失为一个好方法。

单纯的递推之所以无法承受，瓶颈在于 n 的范围太大。

如果把 n 缩小到一个可以忍受的范围呢？

先写一个普通程序计算出所有的 F_x ，然后保留下所有项数是 100000 倍数的 F_x，将它计入一个 list 里，这样在实际计算的时候我们就可以从 $F_{ndiv100000*100000}$ 开始递推，复杂度仅为 O(100000) 。

小结：

“不管白猫黑猫, 捉到老鼠就是好猫。”在这个角度上看，算法一不失为一个好方法。

算法二：

数列是优美的。虽说算法一解决了问题，但是它并没有很好的运用数列本身的特性。**做题并不是应该为了单纯的过题而做题，我们应该精益求精**。

算法一之所以能够成功，是因为它用预处理的方法有效地减少了递推的次数，而减少递推次数的方法不仅仅只有预处理一条路可走。

回想第一章第三部分的内容，如果一个数列是常系数线性递推数列，那么我们可以利用矩阵乘法快速计算出数列的第 N 项。按照上面的思维，**本题的瓶颈在于数列的递推式不是常系数线性的**。但是数列本身不是常系数线性的并不意味着他的子序列不存在着常系数线性递推关系。也许数列中相隔 2 项的元素间存在着常系数线性递推关系，也许是 3 项，也许是更多……倘若**在一个合适的周期 k 下**，元素与元素间存在着线性递推关系，那么我们就可以很好的利用这个关系求解了。

那么是否存在**一个周期并且这个周期适合我们的解题**呢？

再次观察数列的递推式：$g_{x,y} = ((y-1)x^5 + x^3 - xy + 3x + 7y) \mod 9973$ ，这个递推式有两处地方值得注意：

a) 递推式中所有项 y 的指数非 0 即 1，也就是说，如果把 x 看成常数，递推式关于 y 是**线性**的。

b) 递推公式的最后取模了一个素数 9973。

再细细考虑下去这两个特质给我们带来了更多的性质：

令 $M = 9973$，g_n 关于 F_{n-1} 是线性的，并且 $g_{x,y} = g_{x \bmod M,y}$，于是，对于同一个关于 M 的剩余系里的 x，有 $F_{x+M} = (A * f_x + B) \mod M$，也就是说 F_n 是有**周期性**的! 这样我们可以用 O(M) 的时间算出数 A, B 使得 $F_{x+M} = (A * f_x + B) \mod M, \{$ 可以特别选取 $x \mod M = 0\}$，继而算出 F_{kM}，最后算出 F_N。

证明. 令 $U_i = (j^5 - j + 7) \mod M, V_i = (-j^5 + j^3 + 3 * j) \mod M$. 其中 $j = i \mod M + 1$。这样一来就有:

$$F_{i+1} = U_i * F_i + V_i$$

故：

$$F_{KM+1} = (U_{KM+1} * F_{KM} + V_{KM+1}) \mod M$$

$$F_{KM+2} = (U_{KM+2} * F_{KM+1} + V_{KM+2}) \mod M$$

$$F_{KM+3} = (U_{KM+3} * F_{KM+2} + V_{KM+3}) \mod M$$

$$\cdots$$

$$F_{(K+1)M} = (U_{(K+1)M} * F_{(K+1)M-1} + V_{(K+1)M}) \mod M$$

所以：

$$F_{(K+1)M} = (U_{(K+1)M} * F_{(K+1)M-1} + V_{(K+1)M}) \mod M$$

$$= (U_{(K+1)M} * (U_{(K+1)M-1} * F_{(K+1)M-2} + V_{(K+1)M-1}) + V_{(K+1)M}) \mod M$$

$$= U_{(K+1)M} * U_{(K+1)M-1} * F_{(K+1)M-2} + (U_{(K+1)M} * V_{(K+1)M-1} + V_{(K+1)M}) \mod M$$

$$\cdots$$

最后有：

$$F_{(K+1)M} = (A * F_{KM} + B) Mod M$$

其中：

$$A = \prod_{i=KM+1}^{(K+1)M} U_i$$

$$B=\sum_{i=KM+1}^{(K+1)M}(V_i\prod_{j=KM+1}^{i-1}U_j)$$

因为对于 k 不同的取值 k1,k2，

$$U_{k1*M+i}=U_{(k1*M+i)\mod M}=U_i=U_{k2*M+i}$$

$$V_{k1*M+i}=V_{(k1*M+i)\mod M}=V_i=V_{k2*M+i}$$

$$i\in[1,M],i\in N^*$$

故 A,B 为定值。

□

这个算法计算 A,B 需要的复杂度为 $O(M)$，计算 F_{KM} 的复杂度为 $O(K)$ 即 $O(NdivM)$，最后计算 F_N 复杂度为 $O(1)$，故总的复杂度为 $O(NdivM)$。

算法二的再优化：

在本题中 $O(NdivM)$ 就是 $O(M)$ 级别的，算法二已经是一个非常优秀的算法，但是如果 N 的范围进一步扩大的话，算法二也很快将难以承受。

能不能再优化？

既然这个数列在 M 这个周期下数列的递推关系是常系数线性的，那么我们就可以充分利用第一章讲到的方法快速计算。

将 F_{KM} 看成一个新的数列，即：定义数列 C_N，满足 $C_N=F_{NM}$。观察数列 C_N：

$C_0=0$

$C_1=A*C_0+B=(B*1)\mod M$

$C_2=A*C_1+B=(A*B+B=B*(A+1))\mod M$

$C_3=A*C_2+B=(A*B*(A+1)+B=B*(A^2+A+1))\mod M$

...

$C_N=A*C_{N-1}=(A*B*(A^{N-2}+A^{N-1}+\ldots+1)+B=B*(A^{N-1}+A^{N-2}+\ldots+1))\mod M$

故 $C_N=(B*(1-A^N)/(1-A))\mod M$[2]

[2]运用等比数列求和公式，计算可知 A 不为 1

这时，如果我们单纯的计算 $B*(1-A^N)/(1-A)$ 再取模，会发现需要高精度运算，复杂度会变高很多，注意到 $M=9973$ **是一个质数**，故我们可以直接计算 $B*(1-A^N)$，用 3.1 节讲到的快速幂运算得出 $(1-A^N)*B \mod M$，再枚举答案 k 看 $k*(1-A) \mod M$ 与之前计算的值是否相符即可。

当然最后一步还可以用 ExtendedGcd 在 $O(log_2M)$ 的时间内解决：

令 $x=(1-A^N)*B \mod M$，则我们要求的是 $k\in[0,m-1]$ 使 $k*(1-A) \mod M=x$：

这相当于求 $k\in[0,m-1]$ 使 $k*(1-A)+tm=x$。

先用 ExtendedGcd 求 $k*(1-A)+tm=1$，那么 $k*x \mod M$ 即是所求。

能用上述方法的原因是 $M=9973$ **是一个质数**，这导致有且只有一个 k 满足要求：

证明. 先用反证法证明不可能有超过 1 个数均满足要求。

如果有两个数 $k_1,k_2\in[0,m-1]$ 均满足要求，那么就有：

$k_1*(1-A)=k_2*(1-A)(\mod M)$

那么 $(1-A)(k_1-k_2)=0(\mod M)$

由于 $|1-A|<m, 1-A\neq 0, |k_1-k_2|<m$

因此 $k_1-k_2=0$ 即 $k_1=k_2$，矛盾。

故不存在两个数同时满足要求。

所以对于 $k\in[0,m-1]$，$k*(1-A) \mod M$ 对应于 M 个不同的值，即对于每个数 $x\in[0,m-1]$，均有一个 k 与之对应。 □

至此问题得到圆满的解决，时间复杂度为 $O(M+log_2NDivM)$。

2.2　例二 Abs 数列

问题描述：

数列 F_n 满足：

$F_1=A, 1\leqslant A\leqslant 10^{18}$

$F_2 = B, 1 \leqslant B \leqslant 10^{18}$

$F_n = |F_{n-1} - F_{n-2}|, n > 2$

给定 n，求解 F_n。[3]

数据规模：

$3 \leqslant n \leqslant 10^{18}$

分析：

一看到数据规模就知道我们肯定不能通过产生这个数列的前 n 项来求解。而由于递推公式中绝对值符号的存在，导致不同时刻递推公式的形态可能不同，有可能是 $F_n = F_{n-1} - F_{n-2}$，有可能是 $F_n = F_{n-2} - F_{n-1}$，于是我们也不能通过矩阵乘法来解决问题。

既然对着枯燥的式子我们难以产生灵感，那么不妨换一个角度，随机选取几组 A,B，观察数列的前几项，看看能不能找到一些有用的东西：

$A = 0, B = 1 : \{0, 1, 1, 0, 1, 1, 0, 1, 1, \ldots\}$

$A = 1, B = 1 : \{1, 1, 0, 1, 1, 0, 1, 1, 0, \ldots\}$

$A = 25, B = 5 : \{25, 5, 20, 15, 5, 10, 5, 5, 0, 5, 5, 0 \ldots\}$

$A = 65, B = 26 : \{65, 26, 39, 13, 26, 13, 13, 0, 13, 13, 0 \ldots\}$

$A = 23, B = 7 : \{23, 7, 16, 9, 7, 2, 5, 3, 2, 1, 1, 0, 1, 1, 0 \ldots\}$

$\ldots$

容易发现，这些数列有一个非常明显的共同点：所有数列进行到最后都出现了一个长度为 3 周期。

继续观察这些数列，周期里面不为 0 的数 x 各不相同，有 1，有 5，有 13，是什么造成了他们的不同？

把数列与 A,B 联系起来，不难发现：$1 = \gcd(1, 1)$，$5 = \gcd(25, 5)$，$13 = \gcd(65, 26)$，$1 = \gcd(23, 7) \ldots$[4]。

$x = \gcd(A, B)$ ？

[3]经典题

[4]$A = 0, B = 1$ 时是特殊情况，实际上如果我们把这个数列忽略掉第一项看作 $A = 1, B = 1$ 的话一样满足这个条件

这么多的偶然说明事情一定有着它的必然性，事实上我们的猜想也可以得到证明：

证明. 首先用数学归纳法证明这个数列中的所有元素都是 A,B 的最大公约数的倍数：

记 $t = \gcd(A, B)$ 。

1) 当 $n = 1, 2$ 时，$F_1 = A, F_2 = B$ ，因为 $t = \gcd(A, B)$ ，显然成立。

2) 假设当 $n = k - 1, k$ 时，结论成立，即：

$$F_{k-1} = u \cdot t, F_k = v \cdot t, u, v \in N^*$$

那么

$$F_{k+1} = |F_k - F_{k-1}| = |u \cdot t - v \cdot t| = |u - v| \cdot t$$

因为

$$|u - v| \in N^*$$

所以 F_{k+1} 是依然 t 的倍数，即当 $n = k, k + 1$ 时猜想依然成立。

根据 1) 和 2)，可知猜想对任何 $n \in N^*$ 都成立。

再证数列一定会在某处出现 xt,xt :

在没有出现 xt,xt 前这个数列的趋势是递减的。

证明. 因为

$$F_{k+3} < \max(F_{k+1}, F_{k+2})$$

$$F_{k+4} < \max(F_{k+2}, F_{k+3})$$

$$F_{k+4} < \max(F_{k+2}, \max(F_{k+2}, F_{k+1}))$$

$$F_{k+4} < \max(F_{k+1}, F_{k+2})$$

所以

$$\max(F_{k+3}, F_{k+4}) < \max(F_{k+1}, F_{k+2})$$

□

而这个数列元素的最小值为 0，因此数列进行到后面递减的趋势一定会停止，故数列一定会出现 xt, xt

最后用反证证明 $x = 1$：如果 $x \neq 1$，那么数列的所有元素都是 xt 的倍数，又因为 $xt > t$，所以 t 不可能为 A,B 的最大公约数，矛盾，故 $x = 1$。

□

费了这么半天得出 $x = \gcd(A, B)$ 这个结论，我们得到了些什么？

观察证明的过程，再看看证明的命题，**仔细联想**，这与我们了解的求最大公约数的**辗转相除法**极为相似！那么**再联系证明的过程**我们可以发现，数列的前几项应该也与辗转相除法的过程有关。

把数列分成若干**阶段**来看[5]：除去最后出现周期的阶段，每个阶段的开始是 x, y，而结尾是 $y, x \mod y$。

每个阶段具体的细节：

记 $delta = x - y$，那么这段数列的发展过程如下：

$x, x-d, d, x-2d, x-3d, d, x-4d, x-5d, d, \ldots$[6]

又是由若干以 d 为结尾的 3 元素小段组成的。

分析到这，我们就可以构造出算法 5 了：

算法中用 f(x,y,left) 表示当前数列的前两项为 x,y，求第 left 个位置的值。显然我们要求的答案就是 f(A,B,N)

该算法的复杂度分析与辗转相除法的复杂度分析相似，同为 $O(log_2 N)$。

2.3　小结

在例一中我们通过找到递推公式的**周期性**将问题归结为了常系数线性递推数列、结合快速幂运算高效地解决了问题，而在例二中发现数列最后出现**周期性**这一事实展开分析，通过联想、类比辗转相除法，将问题化归、分段考虑最后高效地解决了问题。二者都从**周期性**上找到了突破口，缺少它我们根本无法展开深入的分析，**周期性**是解决这类问题的源头。

[5]为方便起见，如果 $A \leqslant B$，那么去掉数列的第一项，变成 B，$B - A$，这样每个阶段的开始都是 $x > y$ 的形式

[6]第二项 x-d 就是 y

Algorithm 5 calculate $f(x, y, left)$

```
function F(x,y,left)
    if left = 1 then return X
    end if
    if left = 2 then return Y
    end if
    if x ⩽ y then return F(y, y − x, left − 1)
    end if
    Delta ← x − y, Num ← x div Delta
    if even number Num then
        Num ← Num − 1
    end if
    Len ← Num + 1 + (Num + 1)div2
    if left > Len then return F(X − Delta ∗ Num, Delta, Left − Len + 2)
    else
        if Left mod 3 = 0 then return Delta
        else return X − Delta ∗ (Left − Leftdiv3 − 1)
        end if
    end if
end function
```

第三章　多元化方法化归解决数列问题

现在的信息学竞赛中的题目越来越倾向于考察学生的综合能力，而不是一味的套用。常常在一道题中出现多个模型的嵌套，对于数列问题来说也不例外。这就要求我们更加深入地分析，不断挖掘问题的本质，抽丝剥茧，层层深入，充分利用所学过的知识。

3.1　例三 生成树计数

问题描述：

最近，小栋在无向连通图的生成树个数计算方面有了惊人的进展，他发现：

- n 个结点的环的生成树个数为 n 。
- n 个结点的完全图的生成树个数为 n^{n-2}。

这两个发现让小栋欣喜若狂，由此更加坚定了他继续计算生成树个数的想法，他要计算出各种各样图的生成树数目。

一天，小栋和同学聚会，大家围坐在一张大圆桌周围。小栋看了看，马上想到了生成树问题。如果把每个同学看成一个结点，邻座（结点间距离为 1）的同学间连一条边，就变成了一个环。可是，小栋对环的计数已经十分娴熟且不再感兴趣。于是，小栋又把图变了一下：不仅把邻座的同学之间连一条边，还把相隔一个座位（结点间距离为 2）的同学之间也连一条边，将结点间有边直接相连的这两种情况统称为有边相连，如图 3.1 所示。

小栋以前没有计算过这类图的生成树个数，但是，他想起了老师讲过的计算任意图的生成树个数的一种通用方法：构造一个 $n \times n$ 的矩阵 $A = \{a_ij\}$

$$\text{其中} a_{ij} = \begin{cases} d_i & i = j \\ -1 & i \text{ 与 } j \text{ 有边相连} \\ 0 & otherwise \end{cases}$$

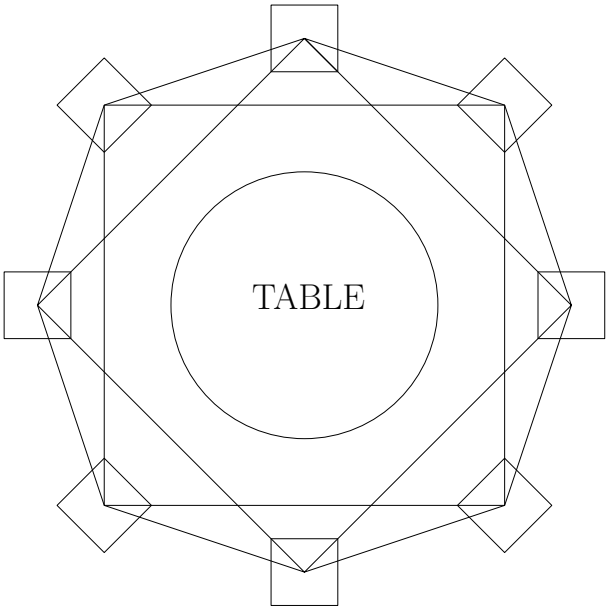

图 3.1:

其中 d_i 表示结点 i 的度数。

与图 3.1 相应的 A 矩阵如下所示。为了计算图 3.1 所对应的生成数的个数，只要去掉矩阵 A 的最后一行和最后一列，得到一个 $(n-1)\times(n-1)$ 的矩阵 B，计算出矩阵 B 的行列式的值便可得到图 3.1 的生成树的个数。

$$\mathbf{A}=\begin{bmatrix} 4 & -1 & -1 & 0 & 0 & 0 & -1 & -1 \\ -1 & 4 & -1 & -1 & 0 & 0 & 0 & -1 \\ -1 & -1 & 4 & -1 & -1 & 0 & 0 & 0 \\ 0 & -1 & -1 & 4 & -1 & -1 & 0 & 0 \\ 0 & 0 & -1 & -1 & 4 & -1 & -1 & 0 \\ 0 & 0 & 0 & -1 & -1 & 4 & -1 & -1 \\ -1 & 0 & 0 & 0 & -1 & -1 & 4 & -1 \\ -1 & -1 & 0 & 0 & 0 & -1 & -1 & 4 \end{bmatrix}$$

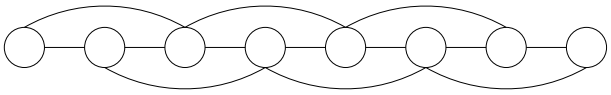

图 3.2:

$$\mathbf{B}=\begin{bmatrix} 4 & -1 & -1 & 0 & 0 & 0 & -1 \\ -1 & 4 & -1 & -1 & 0 & 0 & 0 \\ -1 & -1 & 4 & -1 & -1 & 0 & 0 \\ 0 & -1 & -1 & 4 & -1 & -1 & 0 \\ 0 & 0 & -1 & -1 & 4 & -1 & -1 \\ 0 & 0 & 0 & -1 & -1 & 4 & -1 \\ -1 & 0 & 0 & 0 & -1 & -1 & 4 \end{bmatrix}$$

所以生成树的个数为 $|B| = 3528$。小栋发现利用通用方法，因计算过于复杂而很难算出来，而且用其他方法也难以找到更简便的公式进行计算。于是，他将图做了简化，从一个地方将圆桌断开，这样所有的同学形成了一条链，连接距离为 1 和距离为 2 的点。例如八个点的情形如图 3.2：

这样生成树的总数就减少了很多。小栋不停的思考，一直到聚会结束，终于找到了一种快捷的方法计算出这个图的生成树个数。可是，如果把距离为 3 的点也连起来，小栋就不知道如何快捷计算了。现在，请你帮助小栋计算这类图的生成树的数目。[1]

输入数据：

输入文件中包含两个整数 k,n ，由一个空格分隔。k 表示要将所有距离不超过 k（含 k ）的结点连接起来，n 表示有 n 个结点。

输出数据：

输出文件输出一个整数，表示生成树的个数。由于答案可能比较大，所以你只要输出答案除 65521 的余数即可。

样例输入：

[1]Source:NOI 2007

```
3 5
```

样例输出：

```
75
```

样例说明：

样例对应的图如下：

数据规模和约定：

对于所有的数据 $2 \leqslant k \leqslant n$

数据编号	k 范围	n 范围	数据编号	k 范围	n 范围
1	$k = 2$	$n \leqslant 10$	6	$k \leqslant 5$	$n \leqslant 100$
2	$k = 3$	$n = 5$	7	$k \leqslant 3$	$n \leqslant 2000$
3	$k = 4$	$n \leqslant 10$	8	$k \leqslant 5$	$n \leqslant 10000$
4	$k = 5$	$n = 10$	9	$k \leqslant 3$	$n \leqslant 10^{15}$
5	$k \leqslant 3$	$n \leqslant 100$	10	$k \leqslant 5$	$n \leqslant 10^{15}$

提示：

行列式的一种计算方法，记 $\alpha(P)$ 表示 P 中逆序对的个数，令 B 的行列式

$$|B| = \sum_{P=p_1p_2\ldots p_n \text{是 1 到 } n \text{ 的排列}} (-1)^{\alpha(P)} \prod_{j=1}^{n} b_{i,p_i}$$

如，$B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$，则计算如下：

P	$\alpha(P)$	b_{1,p_1}	b_{2,p_2}	b_{3,p_3}	$(-1)^{\alpha(P)}\prod_{j=1}^{n} b_{i,p_i}$
123	0	1	5	0	0
132	1	1	6	8	-48
213	1	2	4	0	0
231	2	2	6	7	84
312	2	3	4	8	96
321	3	3	5	7	-105

所以 B 的行列式为 $0-48+0+84+96-105=27$。

分析：

首先还是看看找规律能否带给我们这个数列重要的性质。

先看 $k=2$ 的情况，我们用 $f_{n,2}$ 来表示 k=2 时 n 取不同值生成树的个数，他们组成的数列是：

$$1,3,8,21,55,\ldots$$

熟悉 Fibonacci 数列的人都很容易知道，这是 Fibonacci 数列的偶数项的值，即：

$$f_{n,2}=Fibonacci_{2n-2}$$

证明. 用 G_n 表示将 n 个节点按题目要求分成两棵树，并且结点 n-1 与 n 不在一棵树上的情况数。

显然 $f_{2,2}=1, G_2=1$ 而 $f_{n,2}=f_{n-1,2}*2+G_{n-1}$，$G_n=G_{n-1}+f_{n-1,2}$。[2]

将这些递推式联立，再利用数学归纳法，我们就可以得出

$$f_{n,2}=Fibonacci_{2n-2}$$

□

轻易的发现 2 的规律，那么 3 的时候情况如何呢？

$$1,3,16,75,336,1488,\ldots$$

[2]这两个式子可以通过分析连接最后一个节点的边的选取情况可知

很难发现这个数列的特殊性，即使登录互联网上对于数列搜索的权威网站 http://www.research.att.com/~njas/sequences/index.html 这个数列也没有被记载过。

k=3 是如此，k=4,5 亦然。

那该如何寻觅数列的递推公式呢？

仿造 k=2 的证明方法？却发现我们无法像 k=2 时那样理清思路，各种情况错综复杂，实在难于计算。

别急。造成错综复杂的原因是什么？

连的边很容易形成环，无法判断究竟什么情况可行，什么情况不可行。

为什么会形成环？因为连接的两个点他们属于同一连通块。

那么何不增加一维判断连通性？

考虑 N 个节点的情况下，最后 K 个节点的连通性，用最小表示法来表示它们，例如 $N=3$ 的时候有如下 5 种情况[3]：$111, 112, 121, 122, 123$，对于 $N=4$ 有 15 种状态，对于 $N=5$ 则有 52 种。

这样通过 $N-1$ 的情况我们可以推出 N 的情况：考虑最后一个结点往前连边的情况，可以找出一一对应的关系，构造出基本矩阵。

这个预处理可以采用递归，比较易于实现。

接下来计算 $N=k$ 时不同连通性情况下的情况数，作为基底。一样采用递归的方法。

最后进行矩阵连乘。将基底乘以基本矩阵 $N-K$ 次即可。我们的答案是情况 N 下连通性为最后 k 个节点在一个连通块下的情况数。

时间复杂度：

$O(log_2 N \times 52^3)$ [4]

空间复杂度：

$O(52^2)$

[3]数字相同表示他们处于同一连通块

[4]52 是 $K=5$ 时连通性情况数，下同

小结：

到了最后我们发现其实本题并不止一个数列，而答案只是存在于这些数列的其中之一。这些数列按连通性的不同而区分，但是之间又有着关联关系，在解决问题的时候我们先是很好的观察出了 k=2 的情况，之后以递推关系式入手，层层深入，最后运用矩阵连乘的思想很好的解决了题目。

3.2　例四 Maximum

问题描述：

数列 $\{A_n\}$ 满足：

$$A_1 = 1; A_{2i} = A_i; A_{2i+1} = A_i + A_i + 1.$$

现给定 n, 求 $A_1, A_2, \ldots, A_n$ 的最大值。[5]

数据规模：

$1 \leqslant n \leqslant 10^{18}$

分析：

N 的范围如此之大，我们不得不想 O(1) 级别或 O(logN) 级别的算法。观察数列的前几项：

$$1, 1, 2, 1, 3, 2, 3, 1, 4, 3, 5, 2, 5, 3, 4, 1, 5, 4, 7, 3, \ldots$$

乍一看，没有什么规律可循。

看看数列的递推公式 $A_{2i} = A_i; A_{2i+1} = A_i + A_i + 1.$ 与 2 有关，那么这是否意味着它与 2 的幂也有关联呢？

于是如果换种方法，情况就不一样了：

$$1, \{1\}$$

[5]Source:Ural 1396，该数列被称为 Stern's diatomic series 或 $fusc_n$

$$1, 2, \{1\}$$

$$1, 3, 2, 3, \{1\}$$

$$1, 4, 3, 5, 2, 5, 3, 4, \{1\}$$

$$1, 5, 4, 7, 3, 8, 5, 7, 2, 7, 5, 8, 3, 7, 4, 5, \{1\}$$

$$\dots$$

我们发现，第 i 行的结果可以通过第 i-1 行得来：奇数项为上一行，偶数项为相邻的两奇数项相加（最后一个偶数项是最后一个奇数项加 1）。而实际上，我们可以将这个结论推广成更一般的形式：$A_{(2i+1)*2^k} = A_{I*2^k} + A_{(i+1)*2^k}$，这通过数列的性质易证。

除此之外，观察每一行的最大值：

$$\mathbf{1}, \{1\}$$

$$1, \mathbf{2}, \{1\}$$

$$1, \mathbf{3}, 2, 3, \{1\}$$

$$1, 4, 3, \mathbf{5}, 2, 5, 3, 4, \{1\}$$

$$1, 5, 4, 7, 3, \mathbf{8}, 5, 7, 2, 7, 5, 8, 3, 7, 4, 5, \{1\}$$

$$\dots$$

我们还可以发现：第 i 行中最大值为 Fibonacci 的第 $i+1$ 项，并且这个值它与 Fibonacci 的第 i 项相邻：

证明. 首先，当 $i \leqslant 3$ 时，观察数列的前 7 项，显然成立：当 $i > 3$ 时，第 i 行的最大值最大仅可能是第 $i-1$ 行的最大值与第 $i-2$ 行的最大值相加，因为第 $i-1$ 行新产生的数互不相邻（偶数项），原先第 $i-2$ 行互不相邻（奇数项），而第 $i-1$ 行的最大值为 $Fibonacci_i$，第 $i-2$ 行的最大值为 $Fibonacci_{i-1}$，所以第 i 行的最大值为 $Fibonacci_{i+1}$，且 $Fibonacci_{i+1}$ 与 $Fibonacci_i$ 相邻。

□

Algorithm 6 calculate DFSSolve(Count,Left,Right,LeftValue,RightValue)

function DFSSolve(Count,Left,Right,LeftValue,RightValue)
    $Mid \leftarrow (Left + Right)/2$
    $MidValue \leftarrow (LeftValue + RightValue)$;
    if $Mid \leqslant N + 1$ **then**
        $X \leftarrow Fibonacci_{Count+1,LeftValue,MidValue}$
        if $Mid \leqslant N$ **then**
            $X \leftarrow$ Max(X, DFSSolve(Count - 1, Mid, Right, MidValue, RightValue))
        end if
        return X
    else
        return DFSSolve(Count-1,Left,Mid,LeftValue,MidValue)
    end if
end function

Algorithm 7 calculate Main

procedure Main
    Find minimum integer number T satisfied that $2^T > N$
    $Left \leftarrow 2^{t-1}$
    $Right \leftarrow 2^t$
    $Ans \leftarrow Fibonacci_{t,1,1}$　　▷ $Fibonacci_{i,j,k}$ 表示以 j,k 为前两项的广义 Fibonacci 数列第 i 项
    $LeftValue \leftarrow 1$
    $RightValue \leftarrow 1$
    $Ans \leftarrow Max(Ans, \text{DFSSolve}(T, Left, Right, LeftValue, RightValue))$
end procedure

推而广之，在更一般的时候，若两相邻数分别为 $a, b(a < b)$ ，则他们扩展 n 次所生成的数列里的最大值为以 a,b 为前两项的广义 Fibonacci 数列的第 n+1 项，证明原理同上，故略。

回到本题，因此本题可以在 $O(log_2 N)$ 的时空范围内解决：

最后提醒读者：本题的最大值可能会超过 int64 , 所以需要使用无符号 int64 或者实型计算。

小结：

虽然只是一个公式，但是它传递给人的信息却可能远远不止一个公式。看透他的本质再好好加以利用便能为你打造一把成功之匙。

第四章　总结

当今的社会是信息时代的社会，信息技术广泛地运用在各种生产、生活实践中，为各种工程技术提供了强有力的计算工具，为各行各业的发展作出了长足的贡献。长期参加信息学竞赛活动的作者对于信息学的重要性有着深刻的认识，对于信息学有着独到的见解。本文在作者历年参加信息学竞赛活动的基础上，结合作者的所感所想，介绍了处理数列问题的基本知识、常用方法和常见思想。

数列作为特殊的函数，在信息学和实际问题中有着广泛的应用。诸如生产生活中的增长率、降低率，经济活动中的存款利息、分期付款期货贸易、人口增长率等与月份有关的问题常可归纳为数列的问题。

数列问题作为信息学竞赛中一个十分重要的内容，大都考察根据数列的递推公式研究数列通项的性质。本文先提供了与数列问题相关的数学知识，然后给出了快速计算线性递推数列的第 N 项的方法，并在此基础上介绍了合理运用数列“周期性”。作者在文章中提出了解决这类问题的思考方法和常见技巧，主要有深入分析问题本质、寻找数列通项公式，分阶段考虑问题等，并对某些特殊数列的性质和运用做一定的运用和探索。

信息社会的今天，用计算机进行计算，在军事、工程等众多领域必不可少。在诸多算法中，迭代法有着举足轻重的地位，而迭代法又与数列密不可分，因此数列也就显得非常重要。随着素质教育和课程改革的推进，21 世纪的信息学教育越来越重视其综合性和应用性。数列问题以其灵活多变、技巧性强、思维量大、区分度高等特点倍受命题者的青睐，数列的“倩影”越来越多地出现在各级各类信息学竞赛中，命题者往往以探索性、应用性、趣味性题目为载体，考察信息学竞赛选手的知识水平、思维能力与创新意识。

解答数列综合题和应用题既要有坚实的基础知识又要有良好的逻辑思维能力和分析解决问题的能力，我们应充分运用观察、归纳、猜想的手段，建立出有关等差（比）数列、递推数列模型，再结合其他相关知识来解决。

数列问题千变万化，化归不是万能的，但是只要我们能牢牢抓住问题的本质，认真仔细观察，敢于大胆猜想，富有创新精神，发现数列的美，我们就能跨

越艰难险阻，到达成功的彼岸。而这，也是解决其他信息学问题一样需要的。希望读者能够举一反三，多思考，勤分析，开拓思维，将本文所提到的各种思维方法与解题技巧综合地运用到各类题目上，而不局限于数列问题。

希望读者在未来的信息学活动中积极进取，不断进步!

参考文献

[1] http://www.research.att.com/ njas/sequences/index.html The On-Line Encyclopedia of Integer Sequences

[2] http://acm.timus.ru/forum/thread.aspx?space=1&num=1309&id=12571 &upd=633310499364651250, Michael Rybak, 2006

[3] 胡伟栋,“生成树记数解题报告”,2007

感谢

首先感谢我的指导老师陈颖老师。

特别感谢国家集训队教练刘汝佳老师，感谢他对我的论文给出了很多的意见和建议。

感谢长沙雅礼中学的陈丹琦、杭州第二中学的俞华程、广东中山第一中学的顾研，感谢他们给我的论文提供了很好的思路，并且在平常的学习与训练中热心的帮助我，给了我许多启发，告诉了我很多竞赛的技巧。

感谢江苏的陈瑜希、张煜承、陈凯斐，感谢他们提供了 Ural1396 的参考解答。

特别感谢福州一中 2008 届的任欣宇同学，感谢他在各个方面都给了我很大的帮助，在高三繁忙的学习中还帮我的论文梳理文字，并在数学上给了我很大的启发。

感谢福州一中 2007 届的胡伯涛，2008 届的肖汉骏、刘勤、唐健常和张心程，2009 届的朱虹宇，2010 届的李晓潇。