递推关系的建立及在信息学竞赛中的应用

安徽 高寒蕊

(芜湖一中，安徽，241000)

【关键字】递推关系 建立 应用

【摘 要】

世界上的一切事物都在不经意之中变化着，在这纷繁的变幻中，许多现象的变化是有规律可循的。这种规律往往呈现出前因和后果的关系，故我们可以运用递推的思想去研究这些变化。本文着重说明了递推关系的建立，并在此基础上简略介绍求解递推关系的方法。接着，阐明递推关系与动态规划之间的关系，并比较了一般递推关系与动态规划之间的异同，同时举例说明递推关系在竞赛中的应用。在文章的最后，总结出学好递推关系不仅可以提高我们的数学素质，对信息学竞赛更是大有帮助。

目录

【正 文】

1、 引论

瞬息变幻的世界，每一件事物都在随时间的流逝发生着微妙的变化。而在这纷繁的变幻中，许多现象的变化是有规律的，这种规律往往呈现出前因和后果的关系。即是说，某种现象的变化结果与紧靠它前面变化的一个或一些结果紧密关联。所谓“三岁看老”，说的就是这个道理。这一道理也正体现了递推的思想。递推关系几乎在所有的数学分支中都有重要作用，在一切向“更快、更高、更强”看齐的当今信息学奥林匹克竞赛中更因简洁高效而显示出其独具的魅力。在递推关系发挥重要作用的今天，深入研究其性质、特点便成为一件十分必要的事情。本文就将围绕着递推关系的三大基本问题中的如何建立递推关系展开论述，并通过例题说明递推关系在当今信息学竞赛中的应用。

2、 递推关系的定义

相信每个人对递推关系都不陌生，但若要说究竟满足什么样的条件就是递推关系，可能每个人又会有不同的说法。为了更好地研究递推关系，首先让我们明确什么是递推关系。

【定义 1】给定一个数的序列 $H_0,H_1,\ldots,H_n,\ldots$若存在整数 n_0，使当 $n \geq n_0$时，可以用等号(或大于号、小于号)将 H_n与其前面的某些项 $H_n(0 \leq i<n)$联系起来，这样的式子就叫做递推关系。

3、 递推关系的建立

递推关系中存在着三大基本问题：如何建立递推关系，已给的递推关系有何性质，以及如何求解递推关系。如果能弄清楚这三个方面的问题，相信我们对递推关系的认识又会推进一步。由于篇幅所限，本文着重论述三大基本问题之一的如何建立递推关系。

建立递推关系的关键在于寻找第 n 项与前面几项的关系式，以及初始项的值。它不是一种抽象的概念，是需要针对某一具体题目或一类题目而言的。在下文中，我们将对五种典型的递推关系的建立作比较深入具体的讨论。

1．五种典型的递推关系

Ⅰ.Fibonacci 数列

在所有的递推关系中，Fibonacci 数列应该是最为大家所熟悉的。在最基础的程序设计语言 Logo 语言中，就有很多这类的题目。而在较为复杂的 Basic、Pascal、C 语言中，Fibonacci 数列类的题目因为解法相对容易一些，逐渐退出了竞赛的舞台。可是这不等于说 Fibonacci 数列没有研究价值，恰恰相反，一些此类的题目还是能给我们一定的启发的。

Fibonacci 数列的代表问题是由意大利著名数学家 Fibonacci 于 1202 年提出的“兔子繁殖问题”(又称“Fibonacci 问题”)。

问题的提出：有雌雄一对兔子，假定过两个月便可繁殖雌雄各一的一对小兔子。问过 n 个月后共有多少对兔子?

解：设满 x 个月共有兔子 F_x 对，其中当月新生的兔子数目为 N_x 对。第 x-1 个月留下的兔子数目设为 O_x 对。则：

$F_x=N_x+O_x$

而　$O_x=F_{x-1}$，

$N_x=O_{x-1}=F_{x-2}$ (即第 x-2 个月的所有兔子到第 x 个月都有繁殖能力了)

∴　$F_x=F_{x-1}+F_{x-2}$　　边界条件：　$F_0=0$，$F_1=1$

由上面的递推关系可依次得到

$F_2=F_1+F_0=1$，$F_3=F_2+F_1=2$，$F_4=F_3+F_2=3$，$F_5=F_4+F_3=5$，……。

Fabonacci 数列常出现在比较简单的组合计数问题中，例如以前的竞赛中出现的“骨牌覆盖”[1]问题、下文中的『例题 1』等都可以用这种方法来解决。在优选法[2]中，Fibonacci 数列的用处也得到了较好的体现。

Ⅱ.Hanoi 塔问题

问题的提出：Hanoi 塔由 n 个大小不同的圆盘和三根木柱 a,b,c 组成。开始时，这 n 个圆盘由大到小依次套在 a 柱上，如图 1 所示。

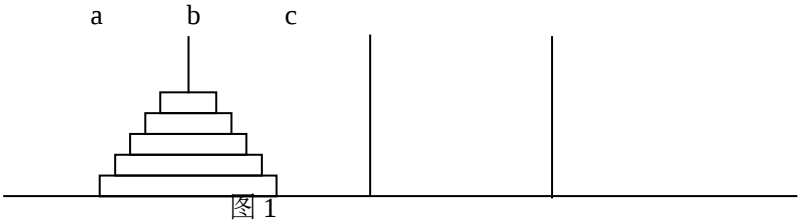


图 1

要求把 a 柱上 n 个圆盘按下述规则移到 c 柱上：

(1)一次只能移一个圆盘；

(2)圆盘只能在三个柱上存放；

(3)在移动过程中，不允许大盘压小盘。

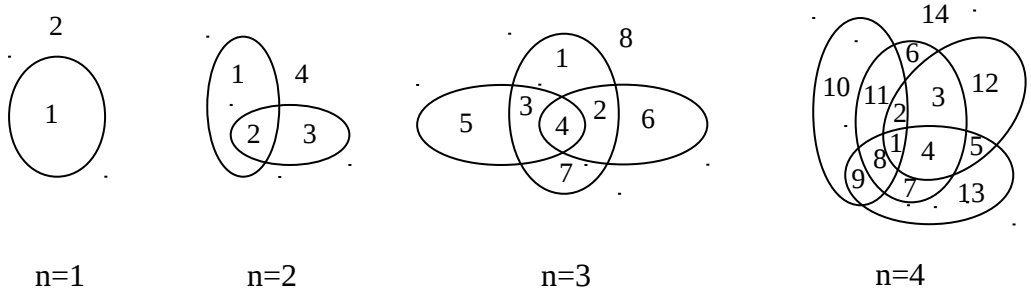
问将这 n 个盘子从 a 柱移动到 c 柱上，总计需要移动多少个盘次?

解：设 h_n 为 n 个盘子从 a 柱移到 c 柱所需移动的盘次。显然，当 n=1 时，只需把 a 柱上的盘子直接移动到 c 柱就可以了，故 $h_1=1$。当 n=2 时，先将 a 柱上面的小盘子移动到 b 柱上去；然后将大盘子从 a 柱移到 c 柱；最后，将 b 柱上的小盘子移到 c 柱上，共记 3 个盘次，故 $h_2=3$。以此类推，当 a 柱上有 $n(n\geq 2)$个盘子时，总是先借助 c 柱把上面的 n-1 个盘子移动到 b 柱上，然后把 a 柱最下面的盘子移动到 c 柱上；再借助 a 柱把 b 柱上的 n-1 个盘子移动到 c 柱上；总共移动 $h_{n-1}+1+h_{n-1}$ 个盘次。

∴$h_n=2h_{n-1}+1$　边界条件：$h_{n-1}=1$

*Ⅲ.*平面分割问题

问题的提出：设有 n 条封闭曲线画在平面上，而任何两条封闭曲线恰好相交于两点，且任何三条封闭曲线不相交于同一点，问这些封闭曲线把平面分割成的区域个数。



n=1　　n=2　　n=3　　n=4

图 2

解：设 a_n 为 n 条封闭曲线把平面分割成的区域个数。 由图 2 可以看出：$a_2-a_1=2$；$a_3-a_2=4$；$a_4-a_3=6$。从这些式子中可以看出 $a_n-a_{n-1}=2(n-1)$。当然，上面的式子只是我们通过观察 4 幅图后得出的结论，它的正确性尚不能保证。下面不妨让我们来试着证明一下。当平面上已有 n-1 条曲线将平面分割成 a_{n-1} 个区域后，第 n-1 条曲线每与曲线相交一次，就会增加一个区域，因为平面上已有了 n-1 条封闭曲线，且第 n 条曲线与已有的每一条闭曲线恰好相交于两点，且不会与任两条曲线交于同一点，故平面上一共增加 2(n-1)个区域，加上已有的 a_{n-1} 个区域，一共有 $a_{n-1}+2(n-1)$个区域。所以本题的递推关系是 $a_n=a_{n-1}+2(n-1)$，边界条件是 $a_1=1$。

平面分割问题是竞赛中经常触及到的一类问题，由于其灵活多变，常常让选手感到棘手，下文中的『例题 2』是另一种平面分割问题，有兴趣的读者不妨自己先试着求一下其中的递推关系。

Ⅳ.Catalan 数

Catalan 数首先是由 Euler 在精确计算对凸 n 边形的不同的对角三角形剖分的个数问题时得到的，它经常出现在组合计数问题中。

问题的提出：在一个凸 n 边形中，通过不相交于 n 边形内部的对角线，把 n 边形拆分成若干三角形，不同的拆分数目用 h_n 表之，h_n 即为 Catalan 数。例如五边形有如下五种拆分方案(图 6-4)，故 $h_5=5$。求对于一个任意的凸 n 边形相应的 h_n。

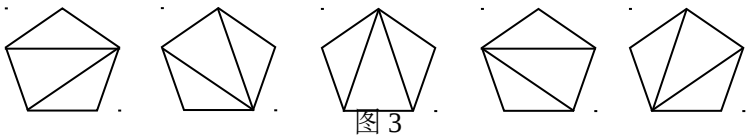


图 3

解：设 C_n 表示凸 n 边形的拆分方案总数。由题目中的要求可知一个凸 n 边形的任意一条边都必然是一个三角形的一条边，边 P_1P_n 也不例外，再根据“不在同一直线上的三点可以确定一个三角形”，只要在 P_2，P_3，……，P_{n-1} 点中找一个点 $P_k(1<k<n)$，与 P_1、P_n 共同构成一个三角形的三个顶点，就将 n 边形分成了三个不相交的部分(如图 3 所示)，我们分别称之为区域①、区域②、区域③，其中区域③必定是一个三角形，区域①是一个凸 k 边形，区域②是一个凸 n-k+1 边形，区域①的拆分方案总数是 C_k，区域②的拆分方案数为 C_{n-k+1}，故包含 $\triangle P_1P_kP_n$ 的 n 边形的拆分方案数为 C_kC_{n-k+1} 种，而 P_k 可以是 P_2，P_3，……，P_{n-1} 种任一点，根据加法原理，凸 n 边形的三角拆分方案总数为 $\sum_{i=2}^{n-1} C_iC_{n-i+1}$，同时考虑到计算的方便，约定边界条件 $C_2=1$。

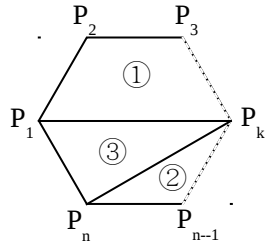


图 4

Catalan 数是比较复杂的递推关系，尤其在竞赛的时候，选手很难在较短的时间里建立起正确的递推关系。当然，Catalan 数类的问题也可以用搜索的方法来完成，但是，搜索的方法与利用递推关系的方法比较起来，不仅效率低，编程复杂度也陡然提高。

*V.*第二类 *Stirling* 数

在五类典型的递推关系中，第二类 Stirling 是最不为大家所熟悉的。也正因为如此，我们有必要先解释一下什么是第二类 Strling 数。

【定义 **2**】**n** 个有区别的球放到 **m** 个相同的盒子中，要求无一空盒，其不同的方案数用 **S(n,m)**表示，称为第二类 **Stirling** 数。

下面就让我们根据定义 3 来推导带两个参数的递推关系——第二类 Stirling 数。

解：设有 n 个不同的球，分别用 $b_1,b_2,\ldots\ldots b_n$ 表示。从中取出一个球 b_n，b_n 的放法有以下两种：

1 b_n 独自占一个盒子；那么剩下的球只能放在 m-1 个盒子中，方案数为 S_2(n-

1,m-1)；

2 b_n与别的球共占一个盒子；那么可以事先将$b_1,b_2,\ldots\ldots b_{n-1}$这 n-1 个球放入 m 个盒子中，然后再将球$b_n$可以放入其中一个盒子中，方案数为$mS_2(n-1,m)$。

综合以上两种情况，可以得出第二类 Stirling 数定理：

【定理】$S_2(n,m)=mS_2(n-1,m)+S_2(n-1,m-1)$　$(n>1,m\geq 1)$

边界条件可以由定义 **2** 推导出：

$S_2(n,0)=0$；$S_2(n,1)=1$；$S_2(n,n)=1$；$S_2(n,k)=0(k>n)$。

第二类 Stirling 数在竞赛中较少出现，但在竞赛中也有一些题目与其类似，甚至更为复杂。读者不妨自己来试着建立其中的递推关系。

小结：通过上面对五种典型的递推关系建立过程的探讨，可知对待递推类的题目，要具体情况具体分析，通过找到某状态与其前面状态的联系，建立相应的递推关系。

2．递推关系的求解方法

求解递推关系最简单易行的方法是递推，递推是迭代算法中一种用若干步可重复的简单运算来描述复杂数学问题的方法，以便于计算机进行处理。它与递推关系的思想完全一致，由边界条件开始往后逐个推算，在一般情况下，效率较高，编程也非常的方便。但是，我们一般只需要求递推关系的第 n 项，而边界条件与第 n 项前面之间的若干项的信息是我们不需要的，如果采用递推的方法来求解的话，第 n 项之前的每一项都必须计算出来，最后才能得到所需要的第 n 项的值。这是递推无法避免的，从而在一定程度上影响了程序的效率。当然在大多数情况下，采用递推的方法还是可行的，在竞赛中，使用递推的方法编程，通常会在时限内出解。

当然，更好的求解方法还有母函数法、迭代归纳法等等，这里就不再一一展开论述了。

例如同样是对于 Fibonacci 数列问题：递推的方法要从F_1，F_2开始逐个推算出F_3，$F_4\ldots\ldots F_n$，而利用母函数的方法，则可以得出公式 $F_n=\dfrac{(\frac{1+\sqrt{5}}{2})^n-(\frac{1-\sqrt{5}}{2})^n}{\sqrt{5}}$，则可直接求出所需要的任意一项。

4、 递推关系的应用

递推关系的应用十分的广泛，其中一个非常重要的应用就是著名的杨辉三角(又称“Pascal 三角”，见图 5)。杨辉三角是根据组合公式$C_n^r=C_{n-1}^r+C_{n-1}^{r-1}$[3]画出来的。很显然，组合公式、杨辉三角都属于递推关系的范围。

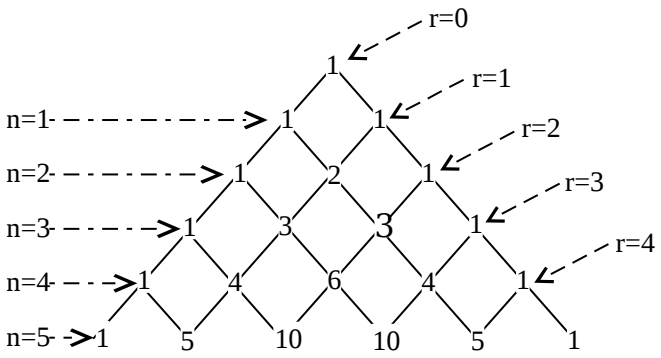


图 5

在今天的信息学竞赛中，递推关系的应用也日趋广泛，下面就让我们从近年来的两道竞赛题中体会递推关系的应用。

『例 1』(1998 蚌埠市竞赛复试第一题)有一只经过训练的蜜蜂只能爬向右侧相邻的蜂房，不能反向爬行。试求出蜜蜂从蜂房 a 爬到蜂房 b 的可能路线数。

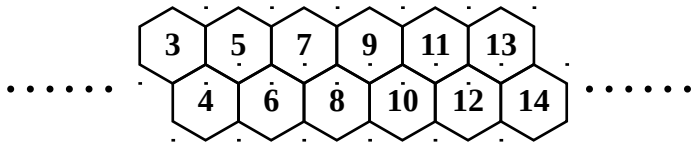


图 6

解：这是一道很典型的 Fibonacci 数列类题目，其中的递推关系很明显。由于"蜜蜂只能爬向右侧相邻的蜂房，不能反向爬行"的限制，决定了蜜蜂到 b 点的路径只能是从 b-1 点或 b-2 点到达的，故 $f_n=f_{n-1}+f_{n-2}$ $(a+2\leq n\leq b)$，边界条件 $f_a=1$，$f_{a+1}=1$。(附有程序 *Pro_1.Pas*)

『例 2』(1998 合肥市竞赛复试第二题)同一平面内的 $n(n\leq 500)$条直线，已知有 $p(p\geq 2)$条直线相交于同一点，则这 n 条直线最多能将平面分割成多少个不同的区域?

解：这道题目与第一部分中的平面分割问题十分相似，不同之处就在于线条的曲直以及是否存在共点线条。由于共点直线的特殊性，我们决定先考虑 p 条相交于一点的直线，然后再考虑剩下的 n-p 条直线。首先可以直接求出 p 条相交于一点的直线将平面划分成的区域数为 2p 个，然后在平面上已经有 $k(k\geq p)$条直线的基础上，加上一条直线，最多可以与 k 条直线相交，而每次相交都会增加一个区域，与最后一条直线相交后，由于直线可以无限延伸，还会再增加一个区域。所以 $f_m=f_{m-1}+m$ $(m>p)$，边界条件在前面已经计算过了，是 $f_p=2p$。虽然这题看上去有两个参数，但是在实际做题中会发现，本题还是属于带一个参数的递推关系。(附有程序 *Pro_2.Pas*)

上面的两道例题之中的递推关系比较明显，说它们属于的递推关系类的试题，相信没有人会质疑。

下面再让我们来看另一道题目。

『例 3』(1999 江苏省组队选拔赛试题第二题)在一个 n×m 的方格中，m 为奇数，放置有 n×m 个数，如图 7：

1643126034-56700260-1-236853400-27-
17407-560-1341242 人

图 7

方格中间的下方有一人，此人可按照五个方向前进但不能越出方格。如图 8 所示：

图 8

人每走过一个方格必须取此方格中的数。要求找到一条从底到顶的路径，使其数相加之和为最大。输出和的最大值。

解：这题在本质上类似于第一题，都是从一个点可以到达的点计算出可以到达一个点的所有可能点，然后从中发掘它们的关系。我们用坐标(x,y)唯一确定一个点，其中(m,n)表示图的右上角，而人的出发点是$\lceil m/2\rceil$,0，受人前进方向的限制，能直接到达点(x,y)的点只有(x+2,y-1)，(x+1,y-1)，(x,y-1)，(x-1,y-1)，(x-2,y-1)。到点(x,y)的路径中和最大的路径必然要从到(x+2,y-1)，(x+1,y-1)，(x,y-1)，(x-1,y-1)，(x-2,y-1)的几条路径中产生，既然要求最优方案，当然要挑一条和最大的路径，关系式如下：$F_{x,y}$= Max{$F_{x+2,y-1}$,$F_{x+1,y-1}$,$F_{x,y-1}$,$F_{x-1,y-1}$,$F_{x-2,y-1}$}+$Num_{x,y}$，其中 $Num_{x,y}$ 表示(x,y) 点上的数字。边界条件为：$F_{\lceil m/2\rceil,0}=0$，$F_{x,0}=-\infty\ (1\le x\le m且x\ne\lceil m/2\rceil)$。(附有程序 *Pro_3.Pas*)

看到上面的题目，肯定有人会说，这不是递推关系的应用，这是动态规划；上面的关系式也不是递推关系，而是动态转移方程。为什么呢？因为关系式中有取最大值运算(Max)，所以它属于动态规划。是吗？递推关系的定义中只要求“用

等号(或大于号、小于号)将 H(n)与其前面的某些项 H(i)(0≤i<n)联系起来”，联系的含义很广，当然也包括用取最大(小)值运算符联系起来。所以我们认为，上面的题目仍可属于递推关系，当然，同时它也属于动态规划。那么递推关系与动态规划之间到底是什么关系呢？我们不妨画个 Venn 图(见图 9)。如果用数学式子表示，就是：A={递推关系}B={动态规划}，B⊂A。通常人们理解的递推关系就是一般递推关系，故认为动态规划与递推关系是两个各自独立的个体。下面让我们通过列表来分析一般递推关系与动态规划之间的异同(见表 1)。

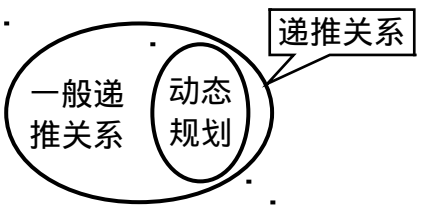


图 9

一般递推关系与动态规划之间的异同对照表

比较项目	一般递推关系	动态规划
有无后效性	无	无
边界条件	一般很明显	一般比较隐蔽，容易被忽视
一般形式	$H_n=a_1\times H_{n-1}+a_2\times H_{n-2}+\ldots+a_r\times H_{n-r}$ 数学性较强	$H_n=Max\{\ldots\}+a$ 数学性相对较弱
是否有阶段	一般不划分阶段	一般有较明显的阶段

表 1

尽管一般递推关系与动态规划之间存在着这样和那样的区别，但是在实际运用中，人们还是经常把它们混为一谈，而且总是把一般递推关系误认为是动态规划。这是因为从上个世纪五六十年代开始被人们发现的动态规划曾在信息学竞赛中掀起了一阵巨浪，直到今天，它仍是信息学竞赛中的重头戏。但是，递推关系的历史源远流长，虽然一般递推关系在今天不如动态规划那么炙手可热，但是它对人的影响是不可忽视的。在 IOI’99 的选拔赛中，就出现过一道《01 统计》[4]，很多人只是先利用公式 $C_n^r=\frac{n!}{r!(n-r)!}$ 求组合数，再对组合数求和，来求 A 类数的个数，导致程序效率不高，其实，这是一道非常典型的递推关系类题目[5]，由于选手平时对一般递推关系的忽视，导致赛场失利。

还有一类博弈问题也利用了递推关系，让我们来看一道关于这方面的例题。

『例 4』(1995《学生计算机世界》)写一个计算机程序，让计算机和人玩纸牌游戏，争取计算机获胜，并显示整个游戏过程。该游戏是：两个选手(计算机一方，人为另一方)比赛：有 n 张(3<n<10000)纸牌，两个选手交替地拿取(计算机先拿)，谁取走最后一张即谁胜。拿取规则为：第一个选手拿走 1 到 n-1 张牌(可拿任意张，但不能拿完)；以后，每个人可取 1 张或 1 张以上，但不得取走大于对方刚取数目的 2 倍(设对方刚取走 b 张，则可取 1 到 2b 张)。

解：这到题目看上去是一道很明显的动态规划试题，以剩余牌数划分阶段，状态 $F_{p,k}$ 表示剩余 p 张牌，且第一人最多可以取 k 张牌的情况是必败点还是必赢点。

状态转移方程是：$F_{p,k}=(p \le k)$ or $F_{p,k-1}$ or not $F_{p-k,2} \times_k$ $(1 \le p \le n,\ 1 \le k<n)$

然后我们可以根据求出的各个状态的情况设计一种取牌方案，使计算机稳赢，当然，如果初始牌数是必败点，那么计算机只能认输。

在牌数不太多的情况下，这种方法效率比较高，但是一旦牌数很大(假设 n 达到极限)，那么需要的空间是 $O(10^5 \times 10^5)$，必然会导致空间不够的问题，这种方法就不可行。

我们不妨把牌数较小的状态画成表来观察(见表 2)，看其中是否存在什么规律。

通过表 2 可以很明显的看出，如果剩余牌数为 2、3、5、8 张的话，无论先取牌的选手取多少张牌(假定不能够一次取完所有的牌)，都必然会输(除非另一个选手不想赢)；像 2、3、5、8 这类数字，我们称之为“必败牌数”。在初始牌数不是必败牌数的情况下，我们要设计一种方案，使每次计算机取过 x 张牌后，剩余的纸牌数大于 2x 张，且为必败牌数。那么究竟那些数字是必败牌数呢？从表中的数字 2、3、5、8 我们猜测 Fibonacci 数列中从 2 开始的数字都是必败牌数，并通过数学证明得证[6]。那么我们就可以根据求出的必败牌数设计方案了。如果想让计算机在每一次取走 x 张牌后剩下的牌数是必败牌数，且大于 2x 张，看来是办不到。例如，初始牌数 20 张，如果一次取走 7 张，使剩余 13 张的话，对手可以一次性将 13 张牌全部取走。那么我们只有再对 7 张牌设计一种方案，保证计算机能取到第 7 张牌，并且计算机最后一次取的牌数小于 13/2 张就可以了，而实现这一步这只需嵌套利用次 Fibonacci 数列就可以了(附有程序 *Pro_4.Pas*)。

n \ k	1	2	3	4	5	6	7	8
1	√	×	×	×	×	×	×	×
2	√	√	×	×	×	×	×	×
3	√	√	√	×	×	×	×	×
4	√	√	√	√	×	×	×	×
5	√	√	√	√	√	×	×	×
6	√	√	√	√	√	√	×	×
7	√	√	√	√	√	√	√	×
8	√	×	×	×	×	×	×	×

注：√表示 True, × 表示 False

表 2

小结：从上面的例题中可以看出，利用一般递推关系解题有时会比动态规划更简单，在动态规划实现起来比较困难的情况下，一般递推关系可能会产生重要作用，这种作用往往表现在直接求解或简化动态规划过程两方面。

5、 总结

通过上文对递推关系的建立和在信息学竞赛中的应用的具体阐述，可知，递推关系不是一种抽象的概念，它是具体的，是需要针对具体题目而言的，因此，我们无法找出一种方法建立出所有的递推关系，只能根据需要解决的题目的具体条件来分析。虽然递推关系的建立没有一个固定的模式可循，但是从总体上来说，都要先找出题目中的重要条件，在这基础上分析某一项与其前面的若干项的关系，然后找出边界条件。递推关系在竞赛中的应用相当的广泛，它包含了几乎每赛必考的动态规划，所以学好递推关系的建立，无论是对提高我们的数学素质，还是今后的竞赛，都大有裨益。因为动态规划更为大家所熟悉，所以

本文着重说明了递推关系中的一般递推关系，希望能给选手一定的启发。

【附 录】[1]

[1] 骨牌覆盖：2×n 的棋盘用 1×2 的骨牌作完全覆盖，求不同的覆盖方式数 C_n

[2] 优选法：设函数 $y=f_x$ 在区间(a,b)上有一单峰极值点，假定为极大点。所谓单峰极值，即只有一个极点 ξ，而且当 x 与 ξ 偏离越大，偏差|f(x)-f(ξ)|也越大。要求用若干次试验找到 ξ 点准确到一定的程度。较优的是实验方法有 0.618 优选法和 Fibonacci 优选法。

[3] 组合公式 $C_n^r = C_{n-1}^r + C_{n-1}^{r-1}$ 的证明：

$$\because C_{n-1}^r = \frac{(n-1)!}{r!(n-r-1)!} \qquad C_{n-1}^{r-1} = \frac{(n-1)!}{(r-1)!(n-r)!}$$

$$\therefore C_{n-1}^r + C_{n-1}^{r-1} = \frac{(n-1)!\times(n-r)+(n-1)!\times r}{r!(n-r)!} = \frac{n!}{r!(n-r)!} = C_n^r$$ 证毕。

[4] 01 序列：近来 IOI 的专家们在进行一项有关二进制数的研究，研究设计的一个统计问题令他们大伤脑筋。问题是这样的：对于一个自然数 n，可以把他转换成对应的二进制数 $\overline{a_k a_{k-1}\cdots\cdots a_1 a_0}$，其中：$n=a_k\times 2^k+a_{k-1}\times 2^{k-1}+\cdots\cdots a_1\times 2^1+a_0$；而且 $a_i=0$ 或 $1(0\le i<k)$，$a_k=1$。如：10=1010；5=101。我们统计一下 a_0~a_k，这 k+1 个数中 0 的个数和 1 的个数。如果在这 k+1 个数中，0 的个数比 1 的个数多，就称为 A 类数。现在的任务是，对于一个给定的 m，求 1~m 中 a 类数的个数。

[5] 有关于《01 统计》的解法可参见徐静同学的解题报告。

[6] 证明：用数学归纳法证明：

1. 由表 2 可以得知：Fibonacci 数 2、3、5、…是必败牌数，且在(1,5)之间的其他牌数都是必赢牌数
2. 假设 Fibonacci 数列 $F_1,F_2\cdots\cdots F_i,F_{i+1}$ 满足在 $(F_1,F_{i+1}]$ 区间，只有 Fibonacci 数才是必败牌数，且其他牌数都是必赢牌数。则我们可证明在 $(F_1,F_{i+2}]$ 区间的牌数也满足上面的性质。证明如下：

设剩下 F_{i+2} 张牌，设这一次，计算机取了 x 张牌，则剩下牌数为 $p=F_{i+2}-x$ 张

(1) 若 $x\ge F_i$，则 $p\le F_{i+1}$，

$\because F_i=F_i+F_{i-1}$ 且 $F_{i-1}\le F_i$

$\therefore F_{i+1}\le 2F_i$

$\therefore p\le 2x$

∴人可以一次将剩下的 p 张牌全部取完

∴计算机一定会输

(2) 若 $1\le x<F_i$，则 $p>F_{i+1}$，

$\because F_{i+2}-F_i=F_{i+1}$ 且 F_{i+1} 是 Fibonacci 数

[1] [3]、[6]的证明是作者自己证明的，若有错误之处，请指正。

∴计算机无法通过一种取牌方案，使计算机在某一次取过少于 $F_{i+1}/2$ 张牌后，剩下 F_{i+1} 张牌

∴当剩下 F_{i+1} 张牌的时候，必然轮到计算机取，且计算机这时不能一次将所有牌取完

∵F_{i+1} 是 Fibonacci 数

∴计算机一定会输

又∵$p\in (F_{i+1},F_{i+2})$,即剩下 p 张牌轮到人取的时候，人一定获胜，

∴p 是必赢牌数

3. 由 1、2 可得结论成立。 证毕。

【参考书目】

1．杨振生 编著 王树禾 审 《组合数学及其算法》 中国科技大学出版社 1997

2．孙淑玲 许胤龙 《组合数学引论》 中国科技大学出版社出版 1999

3．卢开澄 《组合数学》 清华大学出版社 1991

4．吴文虎 王建德 《青少年国际和全国信息学(计算机)奥林匹克竞赛指导——组合数学的算法与程序设计》 清华大学出版社 1997

5．吴文虎 主编 《信息学(计算机)奥林匹克》高级本 北京大学出版社 1992

6．吴文虎 主编 《信息学(计算机)奥林匹克》中级本 北京大学出版社 1992

【程序描述】

Pro_1.Pas

```
program Pro_1;{蜂巢问题，最大范围：目标点与出发点之差不超过 40000}
 type
  Tarr=array[1..10000] of byte;
  Tnum=array[1..4] of ^Tarr;
  Tlen=array[1..4] of word;
 var
     start,finish:longint;
     {出发点和目标点}
     num:Tnum;
     {num[1]…num[3]分别保存到达相邻三个蜂巢的路径数；num[4]是用于交换的临时变量}
     len:Tlen;
     {len[i]表示 num[i]的长度}
```

```
procedure DataInit;{数据初始化}
  var
    i:integer;
  begin
    for i:=1 to 3 do
      begin
        new(num[i]);
        fillchar(num[i]^,sizeof(num[i]^),0);
        num[i]^[1]:=1; len[i]:=1;
      end;
  end;

procedure Add;{高精度加法运算：num[3]=num[1]+num[2]}
  var
    i,carry:word;{carry 是进位数字}
  begin
    carry:=0;
    for i:=1 to len[2] do
      begin
        carry:=carry+num[2]^[i]+num[1]^[i];
        num[3]^[i]:=carry mod 10;
        carry:=carry div 10;
      end;
    len[3]:=i;
    if carry>0 then
      begin
        inc(len[3]);
        num[3]^[i+1]:=carry;
      end;
  end;

procedure Work;{求从出发点到目标点的路径总数}
  var
    i:longint;
  begin
    num[1]^[1]:=1; num[2]^[1]:=1;
    for i:=start+2 to finish do
      begin
        Add;{高精度加法运算：num[3]=num[1]+num[2]}
        num[4]:=num[1];
        num[1]:=num[2];
        num[2]:=num[3];
        num[3]:=num[4];
```

```
        len[4]:=len[1];
        move(len[2],len[1],6);
      end;
  end;

 procedure Out;{输出结果}
  var
   i:integer;
  begin
   for i:=len[2] downto 1 do
    write(num[2]^[i]);
   writeln;
  end;

 begin
  DataInit; {数据初始化}
  write('Input: ');
  readln(start,finish);
  Work;  {求从出发点到目标点的路径总数}
  Out;  {输出结果}
 end.
```

Pro_2.Pas

```
program Pro_2;{第二种平面分割问题}
 var
  n,p,total,i:longint;{total 是区域总数}
 begin
  write('n='); readln(n);
  write('p='); readln(p);
  total:=p*2;
  for i:=p+1 to n do inc(total,i); {也可用 total:=(p+n+1}*(n-p) div 2}
  writeln('Total Area:',total);
 end.
```

Pro_3.Pas

```
program Pro_3;{取数}
 const
  Infns='Pro_3.in';
  Sign=-10000000;{不可到达的点的标志}
 type
  Tarr=array[0..100,1..101] of longint;
 var
  num:^Tarr;{保存每个点上的数字}
  sum:Tarr; {保存到每个点的和最大是多少}
```

```
    n,      {方格的高度}
    m:integer; {方格的宽度}

procedure ReadIn;{读入数据并初始化}
 var
  ni,mi:integer;
 begin
  assign(Input,Infns);
  reset(Input);
  readln(n,m);
  new(num);
  for ni:=n downto 1 do
   for mi:=1 to m do
    begin
     read(num^[ni,mi]);
    end;
  for mi:=1 to m do
   sum[0,mi]:=Sign;  {置不可到达的点的标志}
  sum[0,m div 2+1]:=0;{人的出发点}
  close(Input);
 end;

procedure Work;{找到每个点的和最大的路径的和是多少}
 var
  ni,mi,dir:integer;
  max:longint;
 begin
  for ni:=1 to n do
   for mi:=1 to m do
    begin
     max:=sign;
     for dir:=-2 to 2 do
      if (mi+dir>0) and (mi+dir<=m) and (sum[ni-1,mi+dir]>max)
       then max:=sum[ni-1,mi+dir];
     if max>sign then sum[ni,mi]:=max+num^[ni,mi]
            else sum[ni,mi]:=max;
    end;
 end;

procedure FindMax;{找出和最大的路径的和是多少}
 var
  mi:integer;
  max:longint;
 begin
```

```
    max:=sign;
    for mi:=1 to m do
      if sum[n,mi]>max then max:=sum[n,mi];
    writeln('Max=',max)
  end;

 begin
  ReadIn;  {读入数据并初始化}
  Work;   {找到每个点的和最大的路径的和是多少}
  FindMax; {找出和最大的路径的和是多少}
 end.
```

Pro_4.Pas

```
program Pro_4;{博弈问题}
 var
     Youget,    {人取了多少张牌}
     total:integer; {现在还剩下多少张牌}

 procedure GetYourAnswer(Iget:integer);{读取人这次取多少张牌}
  begin
    repeat
      write('You: '); readln(Youget);
    until (Youget>0) and (Youget<=Iget*2);
    dec(total,Youget); {总牌数相应减少}
    writeln('Remain: ',total);
  end;

 procedure Work(n:integer);{对 n 张牌设计一种取法使计算机能取到第 n 张牌}
     var
       a1,a2,a3:integer;
  begin
    if Youget*2>=n then {是否能够一次取完}
      begin
        writeln('I: ',n); dec(total,n); writeln('Remain: ',total); exit;
      end;
    a1:=0; a2:=1; a3:=1;
    while a3<n do
      begin a1:=a2; a2:=a3; inc(a3,a1); end;
    if n=a3 then begin writeln('You can win!'); halt; end;{必败牌数}
    if ((n-a2)*2>=a2) or (n-a2>Youget*2)
      then Work(n-a2)
      else begin
             writeln('I: ',n-a2);
             dec(total,n-a2); writeln('Remain: ',total);
```

```
        end;
    GetYourAnswer(n-a2); {读取人这次取多少张牌}
    Work(a2-Youget);
  end;

begin                {主程序}
  write('Total card: '); readln(total);
  Youget:=(total-1) div 2;  {通过限定 Youget 给计算机可以取的最大牌数限界}
  Work(total);          {对 total 张牌设计一种取法使计算机能取到最后一张牌}
  writeln('I win!');
end.
```