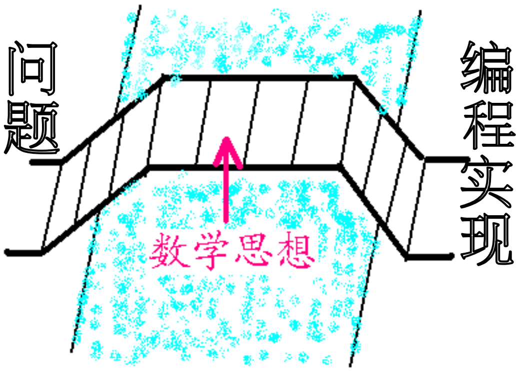
数学思想助你一臂之力

复旦大学附属中学

邵烜程

[关键字]

数学思想、构造算法



[摘要]

数学和计算机原本就是密不可分的学科。有许多计算机编程问题如果不利用数学思想则很难甚至无法达到预期的效果。

如果把问题和编程实现看成是河的两岸，那么数学思想就是连接河两岸的一座桥梁，有了这座桥，从河的一岸到另一岸便不再是件难事了。

有些问题，利用这座桥可以更方便地往返于河两岸，而还有一些问题，如果不利用这座桥，可能根本无法到达河对岸。也就是说，有些问题利用数学思想可以走捷径（例如NOI2002的“荒岛野人”），而还有一些问题，如果不利用数学思想，就根本无法解决（例如NOI2002的“机器人M号”）。我们将从四个方面探讨利用数学思想提高算法效率，简化问题的例子：

一．利用数学思想直接找出解的一般规律。

有些问题，如果直接用动态规划或是图论的方法来解决效率可能会并不理想。这时，我们首先应该想到的是优化，而如果优化无法达到预期的效果，那我们只有重新寻找算法了。于是，我们就试图找出问题的一般规律、或是该问题所用到的一个小问题的一般规律，这样，时间效率将会大大提高。

我们首先来看一个直接找出原问题的一般规律的例子——

例题一　最优分解方案

把正整数 N 分解成若干个互不相等的自然数的和，且使这些自然数的乘积最大。

[输入]

只有一行，包括数 N（$3 \le N \le 1000$）。

[输出]

第一行输出最优分解方案，相邻两数之间用单个空隔隔开；第二行输出最大的乘积。

算法分析

设把 N 分解成 m 个互不相等的自然数 a_1，$a_2, \cdots$，a_m，且满足：$a_1 < a_2 < \cdots < a_m$。

直觉告诉我们：要使乘积最大，就要使 a_1，$a_2, \cdots$，a_m 尽量接近，并且 m 尽可能大。不要怀疑自己的感觉，它往往给我们带来了正确的思路，有时甚至是比其它方法简便得多的。而本题正是这样。

让我们先把这个“感觉”表述成数学语言：

先令 $a_i = i+1(i=1,2,\cdots,k)$，这里 k 满足：

$$\sum_{j=1}^{k} a_j \le N < \sum_{j=1}^{k} a_j + (k+2)$$

设 $rest = N - \sum_{j=1}^{k} a_j$，由于 rest<k+2，故若 rest<k+1，则令

$$a_i = a_i + 1(k - rest + 1 \le i \le k)$$

若$rest = k+1$，则令$a_i = a_i + 1(1 \le i \le k-1), a_k = a_k + 2$，这样得到的$k$个数$a_i$（$1 \le i \le k$）的乘积有最大值。

通俗一点讲，就是将 a_1，$a_2, \cdots$，a_m 首先依次置为 2，3，4，等等。当已经确定的数之和要达到 N 时，将还剩下平均分在前面一些数上。

下面我们来简单证明这个结论：（设 a_1，$a_2, \cdots$，a_m是最优分解方案）

第一步：不存在 $i(1 \le i < m)$,使得$a_{i+1} - a_i > 2$。

证明：若存在$i(1 \le i < m)$，使得$a_{i+1} - a_i > 2$。则令

$$a_k' = a_k(1 \le k \le m, k \ne i, k \ne i+1), a_i' = a_i + 1, a_{i+1}' = a_{i+1} - 1$$

显然

$$\prod_{k=1}^{m} a_k' > \prod_{k=1}^{m} a_k$$

矛盾。

第二步：不存在正整数对（i，j）（$1 \le i < j < m$），满足$a_{i+1} - a_i = 2$且$a_{j+1} - a_j = 2$。

证明：若存在正整数对（i，j）（$1 \le i < j < m$），满足$a_{i+1} - a_i = 2$且$a_{j+1} - a_j = 2$，则令$a_k' = a_k$（$1 \le k \le m$，$k \ne i$，$k \ne j+1$），$a_i' = a_i + 1$，$a_{j+1}' = a_{j+1} - 1$，则$\prod_{k=1}^{m} a_k' > \prod_{k=1}^{m} a_k$，矛盾。

第三步：$a_1 < 4$。
证明：假设 $a_1 \ge 4$。

首先我们明确一点：若$a_i \ge 5(1 \le i \le m)$，则必存在某个j（$1 \le j \le m$），满足$a_j = a_i - 2$。

若不然，则令

$$a_k' = a_k (1 \le k \le m, k \ne i), a_i' = a_i - 2, a_{m+1} = 2$$

显然，有：

$$\prod_{k=1}^{m+1} a_k' > \prod_{k=1}^{m} a_k$$

矛盾。
因此，有：

$$a_1 = 4$$

由于不存在i（$1 \le i \le m$），使得$a_i = 3$，再结合第一步的证明，有：$a_2 = 6$；由于不存在i（$1 \le i \le m$），使得$a_i = 5$，再结合第一步的证明，有：$a_3 = 8$或者$m = 2$，而无论哪种情况，都产生矛盾。故原命题得证。

第四步：若 $a_1 = 3$ 且存在 $i(1 \le i < m)$ 使得 $a_{i+1} - a_i = 2$，则必有

$$i = m - 1$$

证明：

若存在i（$1 \le i < m-1$），满足$a_{i+1} - a_i = 2$，则由第二步的证明，知：$a_{i+2} = a_{i+1} + 1$，则若令：$a_{m+1}' = 2$，$a_{i+2}' = a_{i+2} - 2$，$a_j' = a_j$（$1 \le j \le m$，$j \ne i+2$），则$\prod_{k=1}^{m+1} a_k' > \prod_{k=1}^{m} a_k$，矛盾。故命题得证。

由以上几步的证明，我们便可确认：我们的感觉是无误的。这样，这道题似乎变得异常的简单，因为我们接着要做的，就只是计算这个最大的乘积的值了，这里用到了高精度计算。

让我们再回顾一下解题过程，对于较原始的动态规划算法，我们觉得里面显然有许多不必要的计算，从而提出：能否直接导出一般规律？通过大胆的猜想和严密的证明，这一点得到了肯定，而算法的时间复杂度也从动态规划的$O(N^2)$，下降到了现在的$O(N)$。而这一切都应该归功于数学思想的大胆和合理的应用。

二．利用数学模型化繁为简。

能够对原题导出数学结论无疑是最直接地运用了数学思想，而在很多情况下，往往没有那么简单。在有些实际应用中，我们需要将原来的实际问题抽象成数学模型，然后加以解决。而在某些程序设计竞赛的试题中，建立数学模型也需要一定的技巧，需要运用一定的数学思想。

下面，我们来考虑一个利用数学思想“建模”的例子——

例题二 三角形灯塔

有一个N行（$0 < N \le 50$）的三角形灯塔，它的第1行有1个灯，第2行有2个灯，…，第N行有N个灯。我们用（i，j）表示从上至下第i行，从左至右第j个灯。

每个灯有明暗两种状态，第i行（$1 \le i < N$）的任一个灯（i，j）的状态由下一行的两个灯（$i+1$，j）和（$i+1$，$j+1$）的状态决定（$1 \le j \le i$）。具体的规则如下表所示：

$(i+1, j)$的状态	$(i+1, j+1)$的状态	(i, j)的状态
暗	亮	亮
亮	暗	亮
暗	暗	暗
亮	亮	暗

请你编一个程序，从已知的 P $(P \ge 0)$个灯的状态出发，推出最底一行 N 个灯的所有可能的状态总数。

[输入]

第一行行首为三角形灯塔的行数 N，从第 2 行开始每行为一个已知状态的编号为(i, j) 的灯的信息 $(1 \le i \le N, 1 \le j \le i)$，即三个由空格隔开的整数：$i, j, k$ ，其中 k 为该灯的状态，由 0，1 表示（0 表示暗，1 表示亮）。输入数据以满足 $i = j = k = 0$的一行结束。

[输出]

若问题无解，则输出“NO ANSWER！”；否则输出可能的状态总数。

算法分析

乍一看，似乎只能用搜索来解决，但搜索的时间效率不尽如人意。因此，我们不得不另觅他路。

我们把注意力集中在灯的亮暗规则上。我们容易发现，如果令$l[i, j]$ 表示灯 (i, j) 的亮暗情况（灯亮时为 1，灯暗时为 0），则我们发觉：

$$l[i, j] = (l[i+1, j]) xor (l[i+1, j+1])$$

我们希望利用上面这个特点，得到一个效率更高的算法。为此，我们不希望使用“xor”运算，如果能用加、减、乘或除号来代替“xor”，那么我们就能够用第 N 行的灯的状态来用表达式表示出所有灯的状态。这不由使我们想到了“二进制的无进位加法”，即：

$$l[i, j] = l[i+1, j] + l[i+1, j+1]$$

为方便起见，这里仅用普通的加号来代表二进制的无进位加法（即 0+1=1，1+0=1，

0+0=0，1+1=0）。

这样，若令

$$x[i]=l[n,i](1\le i\le n)$$

我们就可以把每个 $l[i,j]$ 用 $x[i](1\le i\le n)$ 表示出来，根据所给的已知条件，就可以得到若干个关于 x[1]，x[2]，…，x[n]的方程，构成方程组。我们只需要求出该方程组的解的组数即可。

求这个特殊的方程组的解的组数的大致方法是：

先用高斯消元法把方程的个数减少到一个，如果最后剩下的那个方程中含有 p 个未知数（p 为自然数），则由于任意地取定其中 $p-1$ 个未知数的值，则最后一个未知数的值也随之确定，故得到解的组数为 2^{p-1}。

但在求解过程中必须考虑一些特殊情况。

这样，我们就找到了一个较为高效的算法。

在本题中，我们先是由灯的亮暗所遵循的规律发觉了一个较为一般的递推关系，而下面一步才是非常关键的——把递推关系中的逻辑运算转换成数学运算，这一步，为我们构造方程组这一数学模型提供了极大的方便，使问题迎刃而解。

数学思想在本题的应用，使我们方便地构造出了数论模型，使问题圆满地解决。

三．通过数学分析化未知为已知。

有些构造性地问题本身就是由数学问题衍生而来，但正因为问题的“构造性”，使这类问题的解法让人捉摸不透，很难想到。在这种情况下，我们可以试着先从数学的角度分析问题，若能得出对问题直接有益的结论则是最好，如果不能，我们也可以从分析问题的过程中启发思维，从而巧妙地构造算法。

下面来看一个具体的例子体会一下数学分析对解题的帮助——

例题三 配锁问题

某机要部门安装了电子锁。M 个工作人员每人发一张磁卡，卡上有开锁的密码特征。为了确保安全，规定至少要有 N 个人同时使用各自的磁卡才能将锁打开，并且任意 N 个人在一起都能将锁打开。现在需要你计算一下，电子锁上至少要有多少种特征，每个人的磁卡上至少有几个特征。如果特征的编号用从 1 开始的自然数表示，将每个人的磁卡的特征编号打印出来。要求输出的电子锁的总特征数最少。

为了使问题简单，规定：

$$3\le M\le 7,1\le N\le 4,N\le M$$

[输入]

只有一行，包括两个由空格隔开的正整数 M，N。

[输出]

输出包括 M 行，第 i 行有若干个递增的正整数，表示第 i 个工作人员所持磁卡上的全部特征的编号。

算法分析

初看此题，也许你会觉得毫无思路。“至少要有 N 个人同时使用各自的磁卡才能将锁打开”这个看似极其关键的条件到底蕴涵了哪些性质呢？我们不妨先从数学的角度来考虑这个问题，来看一看能否估计出电子锁上特征数的最小值和每人磁卡上特征数的下界。

我们可以证明：

引理：给 M 个工作人员每人发一张具有开锁密码特征的磁卡，规定至少要有 N 个人同时使用各自的磁卡才能将锁打开，则：电子锁上特征数 $tot \ge C_m^{m-n+1}$，每个人的磁卡上的特征数 $per \ge C_{m-1}^{n-1}$。

证明：由于“至少要有 N 个人同时使用各自磁卡才能将锁打开”，意即任意 N-1 个人在一起，都无法将锁打开，从而必然缺少一种开锁的密码特征 A；并且在其余的 M-（N-1）个人中，任意一人加入到 N-1 个人中，他们就能将锁打开，故**这其余的 M−（N−1）个人必同时拥有密码特征 A**。而容易证明：每 N-1 个人在一起，他们缺少的一种密码特征 A 不能和其他一组 N-1 个人一起缺少的密码特征相同（否则，由于这两组至少有 N 个不同的人，且他们都缺少密码特征 A，故这些至少 N 个人在一起无法将锁打开，矛盾）。从而电子锁上特征数

$$tot \ge C_m^{m-n+1}$$

另外，对于每一个工作人员 T 来说，在其余 M-1 个人中，任选 N-1 个人在一起，都会因为缺少某中特征而无法开锁，而这缺少的特征必须是 T 所具备的。故每个工作人员的磁卡上的特征数 $per \ge C_{m-1}^{n-1}$。

事实上，从后面的分析中，我们会发觉：所求的 tot 和 per 的最小值恰好就是 C_m^{m-n+1} 和 C_{m-1}^{n-1}。

在上面的证明过程中，我们最感兴趣的并不是得到的结论（因为本题要求打

印方案，故 tot 和 per 的值完全可以在打印过程中累加），而是在证明的“过程”中用到的：“每 N-1 个人都缺少一种密码特征，而**每 M-（N-1）=M-N+1 个人都同时具备该密码特征**，并且这个密码特征对从 M 个人中选出不同的 M-N+1 个人的组合来说是不同的”。这句话为我们打开了思路，由此，一个有效算法便应运而生了：

初始时，特征数置为 1，在 M 个人中每选取 M-N+1 个人的组合，就为组合中每个工作人员配备当前特征，并将特征数+1。这样，枚举出了所有的

C_m^{m-n+1} 个组合后，便得出了所有工作人员磁卡上特征的方案了。

在本题中，我们通过数学分析，尽管也得出了结论，但由于本题要求输出所有方案，它的用处并不大，但分析过程中所用到的一个思路却直接导致了算法的形成。如果没有进行数学分析，或者没有考虑到这一思路，本题似乎就很难完成了。

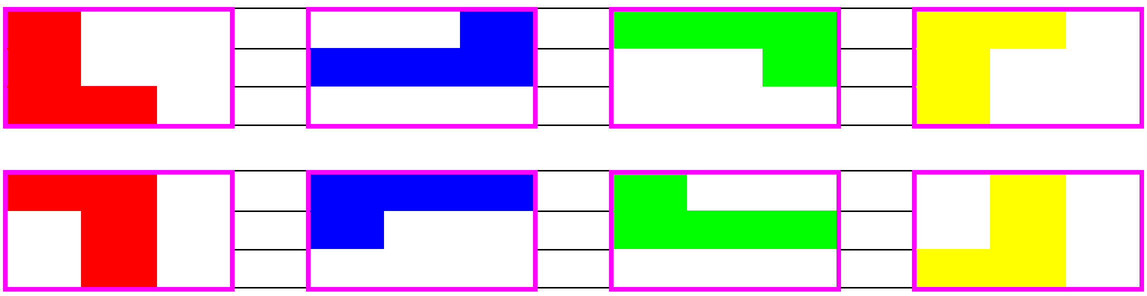
四．利用数学结论优化算法。

在上面的例题中，我们清晰地看到：数学思想的火花孕育了解题的算法。综合上面的三个例题，数学方法、思想的巧妙使用，破解了一个个难题。最后，我们将通过一个并不难的搜索题，体会一下数学在优化算法方面的应用。（读者可以自己比较一下原始算法和优化后的算法在时间效率上的差距）

例题四 骨牌覆盖问题

对于任意一个 $m\times n$ 的矩阵，求 L 形骨牌覆盖后所剩方格数最少的一个方案。

L 形骨牌的 8 种形态



[输入]

输入包括一行，有由空格隔开的正整数 m，n，表示矩阵的大小。

[输出]

输出是一个 $m\times n$ 的矩阵，矩阵中元素的值表示该格所在的骨牌编号（若该格不被骨牌覆盖，则值为 0），骨牌编号为从 1 开始的连续正整数。

算法分析

显然，这道题只有“搜索”，既然是搜索，我们就必然要把注意力集中在优化算法上。对于本题，显然在回溯搜索时，不能让已经“浪费”了的方格数太多，这样再继续搜索下去也是做无用功，于是我们就需要一个“浪费”方格数的上界，也就是最优方案中，最多能覆盖的骨牌数目的下界；而事实上，我们完全可以求出它的确切值。

引理：对于$m \times n$的矩阵（m，$n > 3$），最多能覆盖的骨牌数目

$$number = \begin{cases} \left[\dfrac{m \times n}{4}\right] - 1 & \text{当} m \times n \equiv 4 \pmod 8 \text{时} \\ \left[\dfrac{m \times n}{4}\right] & \text{当} m \times n \neq 4 \pmod 8 \text{时} \end{cases}$$

证明：我们只证明：

$$number \le \begin{cases} \left[\dfrac{m \times n}{4}\right] - 1 & m \times n \equiv 4 \pmod 8 \\ \left[\dfrac{m \times n}{4}\right] & m \times n \neq 4 \pmod 8 \end{cases} \quad (1)$$

至于最大值为何能够取到，用数学归纳法是不难证明的，这里从略。

事实上，我们只需证明(1) 命题对$m \times n \equiv 4 \pmod 8$ 时成立，而其它情况是显然的。为此，我们只需证明：

如果一个$m \times n$的矩阵能用L形骨牌完全覆盖，则所用的L形骨牌数必为偶数。 (2)

为此我们引入L形骨牌的“头部”和“尾部”的概念：即字母“L”底部的两个方格称为“尾部”，另外两个方格称为“头部”。“头部”和“尾部”都类似多米诺骨牌。不妨设n为偶数，则考虑$m \times n$矩阵的每一条水平线，这m条水平线共穿过“L”形骨牌的“头部”或“尾部”的数目总和记为c，一方面，对于每一条水平线，若它穿过奇数个“头部”或“尾部”，则在该水平线上部的其余奇数个方格内要实现多米诺骨牌的完全覆盖是不可能的。因此，每一条水平线必然穿过偶数个“头部”或“尾部”，故c为偶数。另一方面，每个“L”形骨牌恰好被一条水平线切割到其“头部”或“尾部”，从而“L”形骨牌的数目为偶数 (2) 得证。

由于我们计算出了最多能覆盖的骨牌数，搜索过程就大为优化。

算法的大致流程（回溯法）：

自左而右、自上而下搜索每一个方格的覆盖情况，如果可以以该方格为左

上角放置骨牌，则试着放置该骨牌，设置相应的方格被覆盖标志；然后设置该方格不被覆盖标志，搜索下一方格。搜索过程中，一旦已经产生的空格数超出空格数的上界，则回溯。

对比优化前和优化后算法的时效，我们发现，由于加入了槛值，使得优化后的算法减少了很多不必要的搜索，算法的效率自然提高了不少。

本题给了我们这样的启示：数学思想不仅仅能直接提供解题思路和方法，在更多的场合，它只是以一种辅助工具的形式出现的。这就要求我们具备将多种算法，多种思想结合使用的能力。

[结束语]

从上面的几个例子中，我们可窥一斑——数学思想作为沟通问题与编程实现的一座桥梁，有着极其广泛的应用。它不仅可以直接为解题提供思路，如果与其他算法或思想相结合，也能够起到很好的辅助作用。最后，送给大家一句话，希望有所帮助：

在你觉得“山穷水复疑无路”时，不妨用数学的角度重新审视问题，也许就会“柳暗花明又一村”。