论数学策略在信息学问题中的应用

(北京十二中 , 杨江明 , 100071)

【关键字】策略　可扩展性　效率　整数问题

【摘　要】本文研究的是，在信息学竞赛中十分重要，却常常被忽略的数学策略。本文通过分析数学策略中的方程思想、不等式思想及构造法在具体问题中的应用，比较他们同其他策略的优劣，较为详细地介绍了数学策略的效率、应用范围以及可扩展性并总结了在信息学问题中引入数学策略的原因。引申出如何在一般解题过程中应用数学策略。展望了数学策略在今后信息学竞赛中应用的前景。本文所选的例题都是近年来各级信息学竞赛的试题，针对某些题目提出了区别于标准算法的更高效的数学策略解法，具有很强的现实意义。

【目　录】

【正　文】

§1. 数学与策略

数学,是研究现实世界的空间形式和数量关系的科学,是处理客观问题的强有力的工具,几乎在一切自然科学领域中都起着基础性的作用。

策略，是指解决问题所采取的方法。它包括解决各种问题及问题的方方面面的方法。本文讨论的策略，是指利用计算机编程解题时所采取的行之有效的方法，即编程策略。

编写程序解决问题常见的策略有:数学(规律)策略,分治策略,贪心策略,穷举(含搜索)策略等等。

判断某种策略的优劣，通常都从三方面进行考察：

效率：也就是我们所说的算法复杂度。在竞赛中考察程序的复杂度，一般都是考察程序的时间复杂度。当然，时间复杂度同空间复杂度是相互制约的。

应用范围：就是说该策略可以解决哪些类型的题目，是对该策略中“所有”算法所能解决题目总的概括。

可扩展性：针对一个题目所构造的算法，是否可以沿用在其它题目上，如果一个算法可以用在多个题目上，我们就说这种算法的可扩展性大。

我们对下面要研究的数学策略，都从这三方面同其他策略进行比较。

从广义上讲，数学策略包括应用图论的策略，动态规划策略以及应用初等数学手段的策略。图论及动态规划的策略在近年来的比赛中被频频涉及，而初等数学中的方程思想、不等式思想等化简题目、解决问题的手段却没有受到应有的重视。事实上，利用这些基本手段是化简题目的已知条件和建立一个优秀的数学模型必不可少的前提条件。有时能取得意想不到的收获。

本文所讨论的数学策略，是从狭义上，指应用初等数学手段的策略。文章通过分析几道近年来信息学竞赛的题目，比较应用数学策略解决、化简题目与直接运用一般策略在效率上的巨大差异，从而说明数学策略在信息学竞赛中的巨大潜力及在解题上的优势，并尝试总结解决一般问题的应有步骤。

§2. 数学策略在信息学题目中的应用

我们看看我们人类是如何解决具体问题的：人类自身既没有快速的运算系统，也没有大容量的存储系统，我们解题运用的就是我们所擅长的逻辑推理和强大的数学工具，我们有完善的方程理论和不等式思想等等，而这正是计算机所欠缺的。于是，我们尝试让计算机也具有这些优点，贪心策略，A*算法等实际上都是这种有益的尝试。利用人类思考的方式做一些选择，而我们现在所讨论的数学策略，实际上就是这些数学手段的直接运用。【附录 1】

数学策略在信息学中的运用包括两个方面：化简题目和直接解决问题。

应用数学策略化简题目是解决问题必不可少的重要步骤，也是分析题目的基本方法。通过应用数学策略化简题目，发掘题目中的隐含条件，寻求更多的“已知”条件，从而为建立数学模型打下良好的基础。而用数学策略直接解题，其效率更是一般算法所不可企及的。

下面我们分别从方程、不等式及构造法三个方面，对数学策略的应用加以分析。

§2.1 数学策略之方程思想
——化简、解决题目的途径

方程是建立在题目的基础上，对条件的抽象和总结。对于同一题目的不同条件，具有普遍适用性。因此，方程弥补了枚举（包括搜索）策略需要尝试所有情况才能得出结论的缺点。

方程是数学策略中较为重要的一种手段。一般来说，运用方程解决问题，都是运用我们程序较擅长的 n 元一次代数方程组求解，这就涉及到解此类方程组的高斯消元法。

下面讲讲用高斯消元法解一元联立方程组。一元 n 阶线性联立方程组的一般形式为：

$$\begin{cases} a_1\ x_1+a_{12}\ x_2+\cdots+a_{1n}\ x_n=b_1 & (1)\\ a_{2\ 1}x_1+a_{22}\ x_2+\cdots+a_{2n}\ x_n=b_2 & (2)\\ \cdots & \cdots\\ a_{n1}\ x_1+a_{n2}\ x_2+\cdots+a_{nn}\ x_n=b_n & (n) \end{cases}$$

在代数中一般用消元法来解方程组。即：先将第一行乘以一个常数再与其它行相加，以消去其它各行的 x_1 那一项（使 a_{21}，a_{31}，…a_{n1} 为 0）。然后再以新的第二行乘以一个常数并与第 3 行到第 n 行相加，以消去第 3 行到第 n 行上 x_2 的那一项（使 x_2 的系数为 0）。…最后再以新的第（n-1）行乘一个常数并与第 n 行相加，以消去第 n 行上的 x_{n-1} 项。最后得到一个如下形式的三角方程组：

$$\begin{cases} a_{11}x_1+a_{12}x_2+a_{13}x_3+\cdots+a_{1n}x_n=b_1\\ a_{22}`\ x_2+a_{23}x_3+\cdots+a_{2n}`\ x_n=b_2`\\ a_{33}`\ x_3+\cdots+a_{3n}`\ x_n`=b_3`\\ \cdots\\ a_{nn}`\ x_n=b_n` \end{cases}$$

此过程可用图 1 表示。

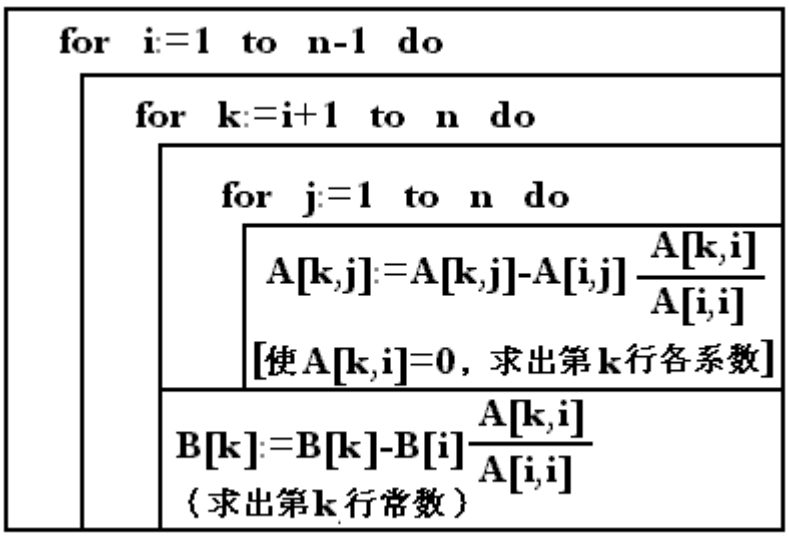

图 1

从此方程组最后一个方程式可以直接求出 $x_n= b_{n`}/a_{nn`}$，然后逐步“回代”，求出 $x_{n-1}, x_{n-2}\cdots$，x_1。

还要考虑一个问题：如果在上面过程中，a_{ii} 为零，则在消元过程中会出现使第 i 行乘以常数 a_{ki}/a_{ii} 而出现无穷大，溢出。例如，本来为了消去第 2 行的 x_1 项，要进行的是：（1）式×a_{21}/a_{11}-（2）式，若 $a_{ii}=0$，则发生溢出错误。必须保证 $a_{ii}\neq 0$。若发生 a_{ii} 为零，可从第 i+1 行到 n 行中找到一个第 m 行，其 $a_{mi}\neq 0$，将此第 m 行与第 i 行对调。如果找不到，则方程无解或无定解。可用图 2 表示。

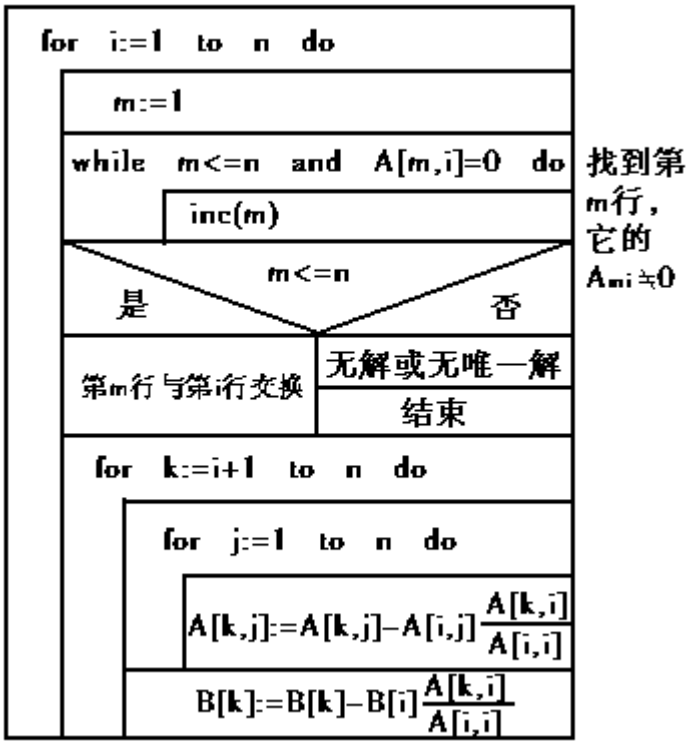

图 2

根据上面介绍的方法，利用图 2 所示的 N—S 图，得到上面列出三角方程组。再使该三角方程组中各 A_{ii} 的值为 1，以得到以下三角方程组：

$$\begin{cases} x_1+a_{12}``x_2+a_{13}``x_3+\cdots+a_{1n}``x_n=b_1 \\ \quad x_2+a_{23}``x_3+\cdots+a_{2n}``x_n=b_2 \\ \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \cdots \\ \quad\quad\quad\quad\quad\quad\quad\quad x_{n-1}+a_{(n-1)n}``x_n=b_{n-1} \\ \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad x_n=b_n`` \end{cases}$$

这样就得到 $x_n=b_n``$，然后回代，求出 x_{n-1}，$\cdots x_1$ 值。画出这部分流程图（图 3）。

上述过程表示为图 2 和图 3。为清晰起见，最好用子程序，一个子程序完成一个功能。

刚刚过去的 IOI'99 为我们留下了许多思考，那我们就由 IOI'99 中的纸牌问题引入吧。

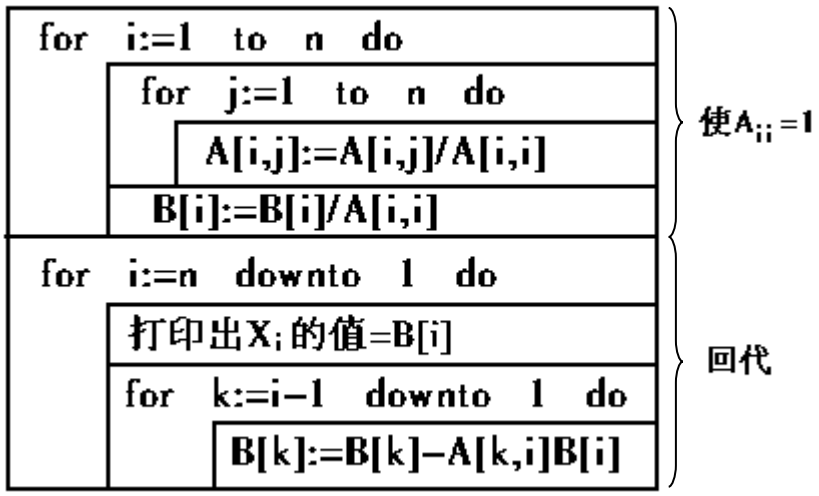

图 3

§2.1.1　方程思想的运用

我们把均分纸牌问题简要描述一下：

【例 1】这是一个均分纸牌的游戏，有 N 列纸牌，每列有纸牌若干张（可能是零张）。纸牌列用从 1 到 N 的整数标号。在移动纸牌时你需要指定一个确定的列 p，和一个确定的数字 m。而后从 p 列上移动 m 张纸牌到每一个相邻的列上。如果 1＜p＜N 的话，则 p 列有两个相邻的列，分别是 p-1 和 p+1；如果 p=1 的话，则只有一个相邻列，其列号为 2；如果 p=N 的话，则只有一个相邻列，其列号为 N-1。

注意，如果 p 列有两个相邻的列，则进行上述移动时，p 列至少要有 2m 张纸牌；如果 p 列只有一个相邻的列，则进行这样的移动时，p 列就需要至少 m 张纸牌。

这个游戏的目的是“均分”所有的纸牌列，使每列都有相同的纸牌数，且用最少的移动达到这一目的。假定有超过一种符合上述要求的移动方法，你只需给出其中一种。[附录中 2]

我们运用数学策略化简题目是为了寻求更好的思路，只有化简题目才能更好地解决题目。但实际上，许多人拿到这道题目时都发懵了，怎样才能“均分”呢？于是，各种学过的算法在头脑中打架：

选手：搜索，你行吗？

搜索：嗯，　10000 步呢，要我嘛，恐怕得等等了。

选手：动态规划，你呢？

动态规划：我？这道题我也没辙。

怎么办？首先，我们对题目进行分析，必须先认识到这样一重关系，第 i 列的纸牌只能向两侧的 i +1 列、 i -1 列移动，而且移动的总牌数是相等的（第 1 列和第 n 列例外）。

其次，只有 i +1 列、 i -1 列可以向第 i 列移动纸牌，而且移到第 i 列的牌数必然等于第 i +1 列、i -1 列移动总牌数的一半。要保证均分，于是对于第 i 列

牌移走的总牌数与移来的总牌数的差值，必然等于首末状态纸牌数的差值。首状态即初始状态，而末状态即均匀状态。于是，对于 N 列纸牌，则共有 N 个未知数即为每列须移走的纸牌数。第 i 列须移走的纸牌数记为 M_i，第 i 列首末状态的差值记为 Δ_i，于是有方程组：

$$\triangle_3\begin{cases} M_2 - M_1 = A - C_1 = \triangle_1 \\ M_1 - 2M_2 + M_3 = A - C_2 = \triangle_2 \\ M_2 - 2M_3 + M_4 = A - C_3 = \triangle_3 \\ M_3 - 2M_4 + M_5 = A - C_4 = \triangle_4 \\ \cdots\cdots\cdots\cdots \\ M_{n-2} - 2M_{n-1} + M_n = A - C_{n-1} = \triangle_{n-1} \\ \triangle n \begin{cases} M_n - M_{n-1} = A - C_n = \triangle n \end{cases} \end{cases}$$

利用高斯消元法，化简得

$$\begin{cases} M_2 - M_1 = \triangle_1 \\ M_3 - M_2 = \triangle_1 + \triangle_2 \\ M_4 - M_3 = \triangle_1 + \triangle_2 + \\ \cdots\cdots\cdots\cdots \\ \cdots\cdots\cdots\cdots \\ M_n - M_{n-1} = \triangle_1 + \triangle_2 + \cdots \end{cases}$$

共有 n 个未知数，n-1 个方程，为一不定方程组。假设 $M_1 = 0$ 代入

可求出对应的一组解 $\{x_1, x_2, x_3 \cdots\cdots x_n\}$，其中 $x_1 = 0$，根据 n 元一次方程组的性质

$\{x_1+t, x_2+t, x_3+t \cdots\cdots x_n+t\}$，t 为整数，也一定为方程组的一组解，代入化简即可证明。

题目要求最小的移动次数，当然移动的纸牌总数也要尽量小。

因为 $x_i \geq 0$，所以若得到最小解为零的一组解，则该解对应移动纸牌总数一定最小。得出结论：

对应每列移动的纸牌数是确定的，且是可求的，而且能够保证均分。

有了均分的保证，剩下的问题就好说了，无论是贪心策略，随机化算法，还是一些算法的综合运用都不成问题。即便用最基本的贪心算法，对于 IOI'99 的测试数据也都能应付自如。

实际上 IOI'99 的这道题目提示我们，数学毕竟是基础，完全脱离数学的算法是没有的，脱离对题目本身的分析，单纯地套用算法是不现实的，而这正是许多选手常犯的毛病。

我们再看一道相关的题目。

【例 2】在物理学中，我们常常对一些复杂的电路问题十分头疼，为了便于分析，我们需要把一些电阻的混连电路，用一个等效电阻来取代。而等效电阻的计算往往是十分繁琐的。于是，我们尝试用程序代替我们完成这项任务。程序需要计算的，是一个纯电阻的混连电路中两点间的总电阻。【附录 3】

【说明】

为了阐述方便，我们建立这样一个模型来描述电路：电路由一个一个结点连接构成，结点就是导线的交点，若两结点间的电路上不存在其它结点，则称这两个结点是两相邻结点。两相邻结点之间只允许有两种情况：

（1） 它们之间是一个已知电阻（如图 4）；

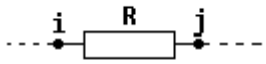

图 4

（2） 它们之间是 x 个已知电阻的纯并联电路（如图 5）；

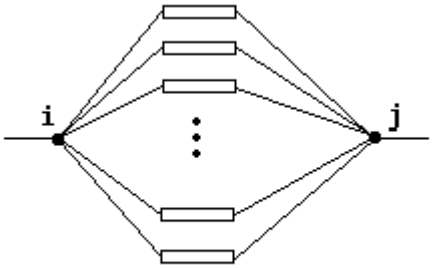

图 5

两相邻结点间总电阻不为零（若为零，则两结点必可以合并成一个结点）。没有孤立的结点。

此模型必然可以描述所有的纯电阻电路。在此基础上我们对此题进行分析。

【输入】

第一行是一个整数 N，表示结点数；

第二行是一个整数 M，表示相邻结点的对数；

第三行有两个数，a 和 b，程序就是要求结点 a 和结点 b 间的总电阻。

以下 M 行每行有三个整数，i,j 和 k（1≤i＜j≤N），表示结点 i 和结点 j 之间连结着大小为 k 的电阻。

【输出】

仅需输出一个数，就是结点 a 和结点 b 间的总电阻。

我们手算解决此类题目，通常都是在结点 a、b 两端接一个外接电源，根据局部电路欧姆定律，测量 a、b 间的电压值和流入的总电流值，从而计算出总电阻。我们用程序解决这道题也可以利用这种手段：加一理想外接电源，给定电压值，解出总电流值，从而求出等价电阻值（如图 6）。对于两相邻结点间存在 x 个电阻的纯并联电路，我们可以在输入的时候将其直接化简，即当读入一个连结结点 i 与 j 的电阻 k 。若结点 i 与 j 之间已记录有总电阻 L,则求出电阻 k 与 L 的并联值，，覆盖掉原来的 L；这样，我们的电路中任意两相邻结点间的电阻已知且不为零。于是，设相邻结点 i 与 j 之间电压为 U_{ij}，电流为 I_{ij}，已知电阻为 R_{ij}，而总电压为 U，总电流为 I，总电阻为 R， R=U/I。

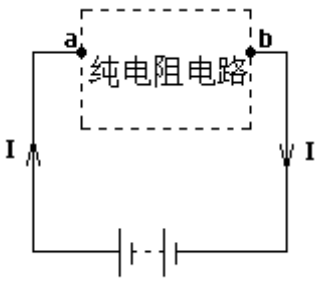

图 6

我们尝试利用现有的知识寻求各个量之间的关系。

在这之前先介绍一下克希荷夫定律。

假设某一电路网络的图 G 有 n 个顶点 m 条边，

$$B=(b_{ij})_{n\times m}$$

$$C=(c_{ij})_{(m-n+1)\times m}$$

分别是它的关联矩阵和回路矩阵。令

$$\begin{pmatrix} i_1 \\ \\ \\ \\ \end{pmatrix} \qquad \begin{pmatrix} v_1 \\ \\ \\ \\ \end{pmatrix}$$

$$I=\begin{matrix} i_2 \\ . \\ . \\ . \\ i_m \end{matrix} \qquad Ve=\begin{matrix} v_2 \\ . \\ . \\ . \\ v_m \end{matrix}$$

分别为这电路的各边电流与各边电压。则我们可得到：

(a)克希荷夫电流定律

对于每一结点，流入该点电流的代数和为零；即

$$\sum_{k=1}^{m} b_{jk} i_k(t) = 0, j = 1,2,\ldots,n$$

或简单地写成：

BI=0。

(b)克希荷夫电压定律

沿着任一回路 C，电压降的代数和为零，即

$$\sum_{k=1}^{m} c_{jk} v_k(t) = 0, j = 1,2,\ldots,m-n+1$$

或写成：

CVe=0 。

根据克希荷夫定律，对于 N 个结点，我们得到 N 个方程。对于所有回路，共有 M-N+1 个本质不同的方程，且 a、b 两点之间的电压降代数和等于电源电压。我们将外电压设为任意 一定值，而任意两相邻点之间的电压可以由其电流表示，$U_{ij}=I_{ij}\cdot R_{ij}$ 。实际上，我们有 M+1 个未知数，而我们也得到了 M+1 个本质不同的方程。根据方程原理，I 可解出，R=U/I，问题得解。

以上两个题目，只是方程思想中一种较为明显的运用。事实上，方程的运用更多的只是一种思想，它是从已知中挖掘更多已知的手段，它更多地应用于竞赛中。当选手拿到题目，对题目分析的过程中，运用方程求出一定解 ，再在此基础上构造其它算法，所以我们称之为方程思想。方程的运用往往都是选手解决题目的一个思维过程，这个思维过程需要选手平时的积累和训练。由于篇幅的限制，仅选出了比较明显运用方程思想的上述两题加以阐述。实际上方程思想在 NOI'96 中的三角形灯塔等问题中，都有较实际的运用。【附录 4】

§2.1.2 运用方程思想同一般策略的比较

运用方程思想解决问题的效率是显而易见的，解一个 n 元一次方程组的算法复杂度，不过是 0（n）级，一旦一道题目可以运用方程思想解出，它的效率必定要比一般算法高很多。上文提到 NOI'96 的三角形灯塔问题，既可用搜索算法解决，又可用方程思想化简解决，而运用方程思想的解法同搜索算法比较，它的高效性也很好地体现出来了。

其次，是应用的范围。方程的另一个显著的优点就是，它可以解许多一般策略无法解决的问题，主要表现在，它并不要求题目一定是整数问题，而搜索策略、动态规划方法恰恰要求问题必须是整数问题，对于非整数问题就显得无能为力了，像例 2 的电阻问题（如图 7）。

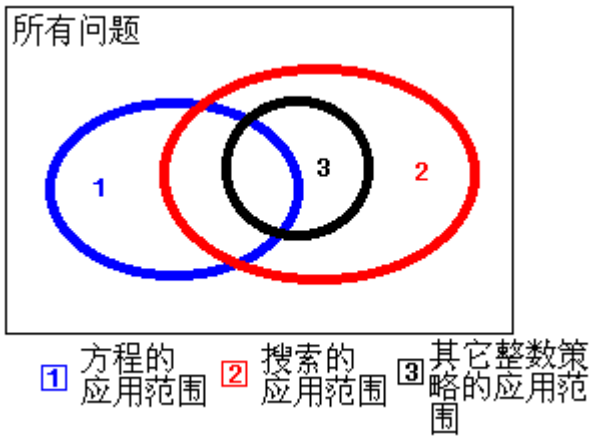

图 7

但是，方程思想也有它自身的缺点，它的运用是有局限性的。对于整数问题，搜索算法相对而言却是“万能”的，例 1 均分纸牌问题，实际上是一道整数问题，我们说的无法解决，不过是指时间上和空间上无法接受而已。而方程思想并不是对于每道题都可以应用的，对于方程思想无法解决的问题，还是要考虑搜索策略。

§2.2　数学策略之不等式
——抽象与具体的桥梁

不等式是表示两个数或两个代数式不相等的算式，在实际应用中，通常用不等式来表示两类数之间或者两个代数式之间存在的普遍关系。应用不等式，因为它不仅仅局限于确定的数字之间的关系，应用不等式也在于它的高效性。我们手工解不等式，运用的是我们的逻辑思维能力，以及利用一些基本不等式和不等式的基本性质。与解不等式相对应的就是枚举。不等式和枚举法实际上是在走两条不同的路，完成同样一个问题。由于程序“不具有”我们的逻辑推理能力，所以我们说枚举法是适合程序设计的。但是，我们一旦将解不等式的方法加入到程序之中，程序的效率将会有一个质的飞跃。

§2.2.1 不等式的应用

【例 3】把正整数 S 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。请编写一程序，由键盘输入 S（3≤S≤1000），求满足条件的分解方案。【附录 5】

【输入要求】 S 由键盘输入。

【输出要求】 ①第一行输出分解方案，相邻两数之间用逗号分开；
②第二行输出乘积（MUL）。

例如：输入：S=10；
输出：2，3，5；
MUL=30

设 $S=a_1+a_2+a_3+\cdots+a_n$　$(1\leqslant a_1<a_2<\cdots<a_n)$

$$P= a_1 \cdot a_2 \cdot a_3 \cdots \cdot a_n$$

我们求的就是 P 的最大值。

根据均值不等式，由于 a_i 为正

$$\frac{a_1+a_2}{2} \geq \sqrt{a_1 a_2}$$

$$\frac{a_1+a_2+a_3}{3} \geq \sqrt[3]{a_1 a_2 a_3}$$

.....

$$\frac{a_1+a_2+a_3+\ldots+a_n}{n} \geq \sqrt[n]{a_1 a_2 a_3 \ldots a_n}$$

即 $$\left(\frac{s}{n}\right)^n \geq p$$

由于 $1 \leqslant a_1 < a_2 < a_3, \cdots < a_n$　　所以 $s/n > 1$。

要使得 p 最大，则须使 $(s/n)^n$ 最大，又∵ $s/n > 1$，则使得 n 最大。同时设法使相邻两数 $a_{i+1}-a_i$ 的差最小。设 $a_1=2$（若 $a_1=1$, 则 a_1 对乘积将失去其作用），求出符合条件的:

$2+3+4+5+\cdots+a_n+x=S$ ($2, 3, \cdots, a_n$ 是连续的自然数，且 $a_n \geqslant x$)。

即将 S 分解成尽可能多的前 n-1 个连续自然数，剩余 x 。X 必小于等于 a_n。因为若 $x > a_n$，则一定可以在 a_n 后面补上 a_{n+1}，分解成的原序列就不是尽可能多的了，矛盾，故 $x \leqslant a_n$。而 x 必应与数列 $\{a_i\}$ 中某数重复，因此必须撤去 x。为保证撤去 x 后各个自然数互不相等，其和还是等于 p 且乘积最大。我们将数 x 尽量平均地加在后几项，并尽可能使得相邻两数的差不超过 2。

1 若 $a_{n+1}=1$, 由于 $1*2\cdots*a_n < 2*3\cdots*(a_n+a_{n+1})$, 因此将 $a_{n+1}=1$ 加到 a_n 上;

2 若 $1 < a_{n+1} < a_n$（a_{n+1} 与 2，…，a_n 中的某一个数相同），我们从 a_n 出发依次向尾部的 a_{n+1} 个数加 1;

3 若 $a_{n+1}= a_n$，a_n 加上 2，其余的 $a_{n+1}-2$ 个数依次加 1。

当然，还必须考虑一个例外情况：当 n=3 或 4 时，只能分解出一个方案：

3=1+2　　MUL=1*2=2

4=1+3　　MUL=1*3=3

简单的问题，简单的思考，简单的程序，一切显得如此轻松。但你是否发现其中蕴涵着的数学的美?

实际上题目的意义不在于题目的本身，它是一种思考。我们设计程序解决此类问题，并不局限在枚举策略上。利用不等式化简题目，应是我们的选择方向而且应当是首选。

我们看一个具体的例子。

【例 4】排序网络是这样一个模型:

有 n 条水平的导线，由上至下分别标号为 1 至 n，导线与导线之间不相交。在任意两导线之间可以搭上“电桥”。每条导线的左端放置一数字，数字沿导线

由左向右同步运行，当遇上一电桥时，比较电桥两端的两根导线上的数字，将其中较大的放到上面的导线上，而将较小的放到下面的导线上，然后继续运行，保证同一时刻最多只有一个电桥。（如图 8）

若一个 n 行的排序网络共包含 m 个电桥，对于左端输入的任意顺序的数列，都能在右端得到一个由大到小的数列，我们称这个排序网络是一个完全排序网络。

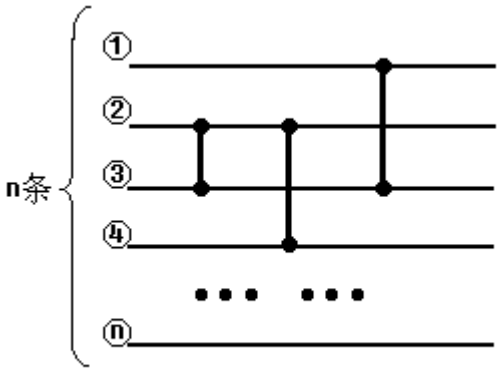

图 8

我们的程序需要做的，就是判断一个排序网络是否是一个完全排序网络。【附录 6】

抛开我们一贯采用的枚举策略，看看我们是怎样运用不等式的性质，极其巧妙而且高效地解决这个问题的吧！

研究这道题，就不能不谈谈排序。排序的本质实际上就是找到一个数列各项之间的关系对应的不等式组。对于这道题，数列是确定的，在网络中运行的过程，实际上就是构造不等式组的过程。分析一个完全排序网络（如图 9）就会发现，本题构造不等式的过程是一个分阶段的过程，每经过一个电桥对应一个阶段，每个阶段都重新排列两个数，每个阶段也都是在对不等式组做一次“修正”。

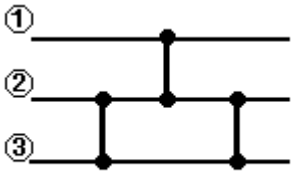

图 9

具体分析图（9），我`们用 A_{ij} 表示经过第 i 个阶段后，第 j 条导线中的数。

阶　段	排　列	不　等　式　组
阶段一	2/3	$A_{1,2}\geq A_{1,3}$　$A_{1,1}$ 与 $A_{1,2}$ 之间　$A_{1,1}$ 与 $A_{1,3}$ 之间不确定
阶段二	1/2	$A_{2,1}\geq A_{2,2}$　$A_{2,1}\geq A_{2,3}$　$A_{2,2}$ 与 $A_{2,3}$ 之间不确定
阶段三	2/3	$A_{2,2}\geq A_{2,3}$　$A_{2,1}\geq A_{2,2}$　$A_{2,1}\geq A_{2,3}$

我们发现，阶段 i 的不等式组，可以由阶段 i-1 的不等式组根据阶段之所进行的重排列情况推导得出。编号为 i 和 j 的导线进行重排列之后，导线 i 和 j 之间建立了不等式关系，而所有关于导线 i 和 j 的其它不等式都需要进行调整。不等式间的推导属于逻辑推导。我们的程序是否可以进行这种逻辑推导呢？答案是：可以。

假设我们已经求得了阶段 i-1 时对应的不等式组，阶段 i 是对导线 a、b(a < b)进行重新排列，首先得到 $A_{ia}\geq A_{ib}$，下面我们要对所有与导线 a,b 有关的不等式进行调整。对于 a,b 分别分五种情况。

一、调整所有关于 a 的不等式。

①原有不等式 $A_{i-1,c}\geq A_{i-1,a}$ （c < a 且 c≠b），$A_{i-1,c}$ 与 $A_{i-1,b}$ 关系不确定
则 $A_{i,a}\geq A_{i,b}$，$A_{i,c}\geq A_{i,b}$，$A_{i,c}$ 与 $A_{i,a}$ 关系不确定

②原有不等式 $A_{i-1,a}\geq A_{i-1,c}$，（a > c 且 c≠b），$A_{i-1,c}$ 与 $A_{i-1,b}$ 关系不确定
则 $A_{i,a}\geq A_{i,b}$，$A_{i,a}\geq A_{i,c}$，$A_{i,b}$ 与 $A_{i,c}$ 关系不确定

③原有不等式 $A_{i-1,a}\geq A_{i-1,c}$，$A_{i-1,b}\geq A_{i-1,c}$，
则 $A_{i,a}\geq A_{i,b}$，$A_{i,a}\geq A_{i,c}$，$A_{i,b}\geq A_{i,c}$

④原有不等式 $A_{i-1,a} \leqslant A_{i-1,c}$，　$A_{i-1,b} \leqslant A_{i-1,c}$，
　则 $A_{i,a} \geqslant A_{i,b}$，$A_{i,a} \leqslant A_{i,c}$，$A_{i,b} \leqslant A_{i,c}$
⑤原有不等式 $A_{i-1,a} \geqslant A_{i-1,b}$ 则不变。

二、调整所有关于 b 的不等式。

①原有不等式 $A_{i-1,c} \geqslant A_{i-1,b}$ （c < b 且 c≠a），$A_{i-1,c}$ 与 $A_{i-1,a}$ 关系不确定
　则 $A_{i,a} \geqslant A_{i,b}$，$A_{i,c} \geqslant A_{i,b}$，$A_{i,c}$ 与 $A_{i,a}$ 关系不确定
②原有不等式 $A_{i-1,b} \geqslant A_{i-1,c}$，（b > c 且 c≠a），$A_{i-1,c}$ 与 $A_{i-1,a}$ 关系不确定
　则 $A_{i,a} \geqslant A_{i,b}$，$A_{i,a} \geqslant A_{i,c}$，$A_{i,b}$ 与 $A_{i,c}$ 关系不确定
③原有不等式 $A_{i-1,a} \geqslant A_{i-1,c}$，$A_{i-1,b} \geqslant A_{i-1,c}$，
　则 $A_{i,a} \geqslant A_{i,b}$，$A_{i,a} \geqslant A_{i,c}$，$A_{i,b} \geqslant A_{i,c}$
④原有不等式 $A_{i-1,a} \leqslant A_{i-1,c}$，　$A_{i-1,b} \leqslant A_{i-1,c}$，
　则 $A_{i,a} \geqslant A_{i,b}$，$A_{i,a} \leqslant A_{i,c}$，$A_{i,b} \leqslant A_{i,c}$，$A_{i-1,a} \geqslant A_{i-1,b}$
⑤原有不等式 $A_{i-1,a} \geqslant A_{i-1,b}$ 则不变。

以上的讨论覆盖了所有可能情况，其实具体编程的时候还可以化简。如果上述推导是正确的，那么，最终我们所求得的不等式组与排序网络原本所对应的不等式组是等价的。于是，我们可以通过分析不等式组的拓朴情况，证明排序网络是否符合要求。

下面对上述推导做出证明。

欲证明推导的正确性，只须对每种情况的正确性做出证明。分析关于 a 的不等式的第一种情况，将其抽象为①原有 a、b、c 三个数，c≥a, c 与 b 关系不确定；②a`=MAX{a,b}，b`=MIN{a,b}，c`=c

欲证明 a`≥b`，c`≥b`，a`，c` 关系不确定。

证明：当 a≥b 时，∵c≥a ∴c≥a≥b
　　a`=a，b`≥b，∵c`=c　∴c`≥a`≥b`
　　当 a<b 时，c≥a；b>a，b、c 关系不确定
　　a`=b，b`=a，c`=c，∴c≥b`，a`≥b`，a`、c` 关系不确定

综上所述，任意情况下，都有：a`≥b`，c`≥b`，而 a`、c` 关系有时不确定（即 a`、c` 关系不确定）。得证。

对于其它情况也可按此法进行证明，由于篇幅限制，从略。

由此可见推导的正确性，从而仅需根据我们得到的不等式，判断其中是否任意两导线都是拓朴有序的，即可证明排序网络是否是完全排序网络。

解决这道题目的意义在于，它证明了一件事：那就是程序设计也可以进行一些“复杂”的逻辑推导。不过，解不等式需要选手对问题进行详细的分析，从而区分各种情况，再设计算法。

§2.2.2 应用不等式同一般策略的比较

将解不等式同程序设计结合起来，本身就是一个极其有意义的事情，让计算机具有一定的逻辑推导能力，从而弥补计算机的不足。

在效率上，解不等式之所以效率高，是因为应用不等式组，用抽象代替一般，省去了枚举各种情况的耗费，解不等式适合人类运算速度“不快”的特点。如果计算机将其运用，其效率自然是可观的。

例 4 的“标准”算法是应用 0、1 理论【附录 7】。我们在 PⅡ350 的机器上拿它和不等式的解法做了一下对比：

解　法	算法复杂度	效　　率
0、1 理论	$0(m \cdot 2^n)$	当 n 达 20 多时，时间上就不可忍受了
不等式	$0(m \cdot n)$	在空间允许的范围内，时间始终在 0.1S 之内

事实证明，不等式是一种极其优秀的解题手段，在可能的情况下应优先考虑，关键还是靠选手的灵活运用。

不等式同方程一样，范围上都不受“整数问题”的限制，这也是不等式的优越性之一。

不过，不等式终究是有一定的应用范围的，更多时候需要对具体问题作具体分析。

§2.3 特殊的问题——构造法
——到达想象力的尽头

用构造法解决问题，首先需要对题目有深刻的了解，然后还要有巧妙的构思。一个问题如果使用构造法解决，无论是从思路上，还是从程序上都已经精简到了极限。因此，可以说，构造法是最高级的算法。

§2.3.1 构造法的应用

构造法不同于不等式与方程，构造法都是一对一的，一个具体的问题对应一个算法，不同问题之间很少有相关联的，因此构造法没有固定的模式可循。

正是因为构造法所具有的这种特殊性质，列举太多的题目是没有什么意义的，在这里我们仅分析一道经典题目，就算是抛砖引玉吧！

【例 5】N 个正整数横向排列，两人轮流从中取数，每次只能取走最左端或者最右端的数，所取数的数值表示这个人这次的得分。所有数都取完时，每个人所得分数之和为这个人的最后得分，最后得分多的人获胜。如果两人得分相同，则规定首先取数的人获胜。【附录 8】

如果游戏开始 N 为偶数，则对首先取数的人来说存在一种必胜的策略。请编一个程序，作为首先取数的人，寻找一种必胜的策略和计算机玩这个游戏。

我们不妨将题目做一下处理，把要处理的对象染上不同的颜色，从而可以区分它们，这样才能够看出其中的一些隐蔽的本质。将要拿的数字染成不同的颜色，像这样：

图 10

实际上，先拿的人有权拿走所有自己想要颜色的数字。也就是说，先拿的人想拿走深色的 4，2，5 这三个数，就可以拿走，具体是这样的：

（1）拿走 4，对手只能拿 7 或 9，而它们都是浅色的；

（2）无论对手怎样拿，都有一个（也仅有一个）深色数字可拿，那么就将它拿走；

（3）对手仍然只能选择浅色数字；

（4）…

这样拿下去，只要先拿的人每次都选择深色数拿，必定能够将所有深色数拿走。

这意味着什么？这意味着先拿的人只要选择两种颜色中数字总和较大的那组数，就可以保证胜利了。

真正在竞赛中运用构造法需要选手的创造力。当然问题的解法是多样的，正所谓条条大路通罗马。构造法解题的数学性极强，要求选手有比较深厚的数学功底，同时更需要有较为敏锐的观察能力和归纳推测能力。

§2.3.2 构造法同其它策略的比较

首先，要把和构造法极容易混淆的贪心策略相区别。其实，构造法同贪心策略的区别也是很明显的。从本质上讲，构造法是最优解算法，而贪心策略是近似解算法。贪心策略是从问题的局部考察，从而做出选择；而构造法是通过对全局的分析，构造出可证明的最优策略。

在效率上，构造法是所有算法中最优的，就连同为数学策略中的不等式、方程以及我们所偏爱的动态规划也自愧弗如。

在解题范围上，构造法运用得极其广泛，但是构造法却没有丝毫可扩展性可言。

构造法作为数学策略中的一部分，虽然它有种种不足，但是它的高效却是十分诱人的，成为许多选手的追求。在解题中，如果想出了构造法的算法，这种算法自然就是首选。

§3.　为什么应用数学策略
——小结数学策略的应用

上文具体讲述了数学策略的应用，下面就解释一下为什么要在信息学竞赛中应用数学策略。这个问题可以从两个方面来回答。

1、 数学策略能够解决这一类问题的难点。

首先，数学策略能跨跃整数问题的界限，拓展了我们的解题范围，从而解决许多一般策略无法解决的问题。

其次，数学策略具有极高的效率，而效率正是我们所期望的。

2、 数学策略避开了枚举策略的主要缺陷。

数学策略与枚举策略是站在不同的高度看问题，为什么这样说呢？

假设我们分别用数学策略和枚举策略处理同一问题，数学策略就是将这个问题先抽象，列出不等式、方程组，然后运用数学手段寻求问题中存在的普遍关系；而枚举法恰恰相反，它是对问题的所有情况先做分析，然后从中提出共性的东西，总结出问题普遍存在的关系（如图 11）。我们应用数学策略就是要弥补枚举策略在列举各种情况的过程中所做的过多耗费。

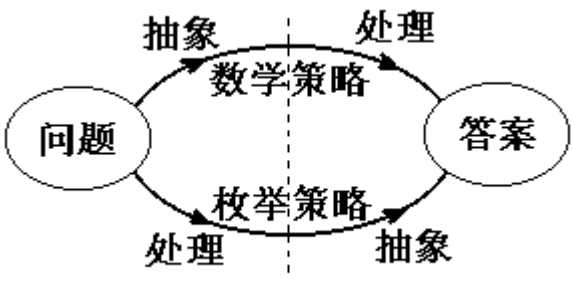

图 11

正是因为数学策略与枚举策略相互补充，所以经常用数学策略来解决枚举策略无法胜任的工作。

[跋语]

数学策略是人类智悲的结晶，能运用好数学策略是对选手的基本要求。数学策略有优点，也有缺点。其实，每种策略所能处理问题的规模都是有限的，在编程解决的时候，必须明确问题规模，从而谋划适当的策略，构造相对应的算法。数学策略之所以高效，是因为数学策略比一般的枚举策略更具有针对性。通常而言，算法针对性越高，其效率也越高（如构造法），而扩展性也就越低，且构造起来也越难。对于所有策略，我们总结可扩展性与效率、构造难易与效率基本上都呈反比例关系（如图 12、图 13）。我们解决问题，就是要从中选出一个最佳平衡点。

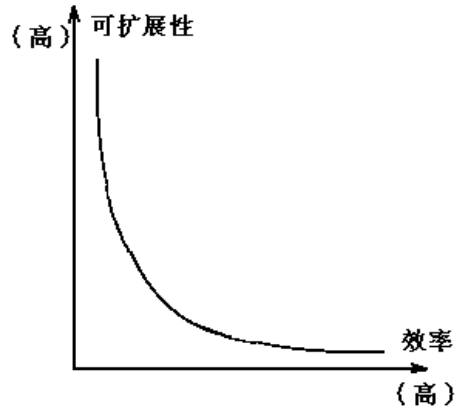

图 12

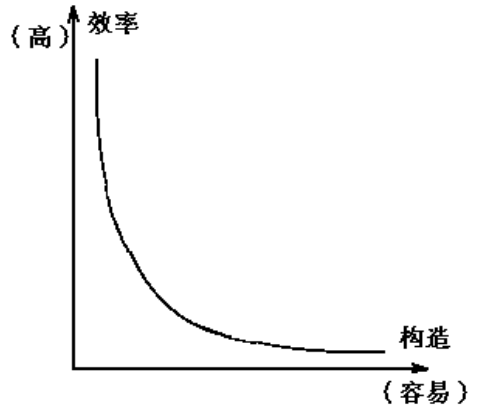

图 13

数学家波利亚的《怎样解题表》中曾经这样说：“如果你不能解决这个问题，你能否解决问题的一部分？”当我们拿到一个陌生的问题时，我们不妨先尝试数学策略，哪怕数学策略不能完全解决问题，我们也可以化简其中的一部分，从而为我们再次选择其它策略铺平道路（如例 1）。正因为数学策略的优越性，所以我们说，尝试数学策略原本应是处理问题的第一步。

数学，不但是数学策略的基础，也是整个信息学的基础。目前信息学竞赛题目日趋复杂，单纯套用算法的时代一去不复返了。当前，更多的考察的是算法与算法间的综合应用，而数学作为信息学竞赛中密不可分的一部分，它与题目的联系也越密切，将数学应用同信息学解题相结合，恐怕也是一种趋势吧！

数学是一种美。有一位科学家曾经说过：如果让我在真和美之间选择，我愿选择美。当然，真和美在很多问题上是统一的，尤其在数学这样的学科中。在我们编程之中，如果能够更好地了解掌握和应用数学，一定会达到真和美的和谐统一。

就以此作为这篇论文的结束吧！

【附　录】

1、我们这里所说的数学策略，也是人工智能的范畴，属于人工智能中的专家系统。专家系统，简要地说，就是使计算机尽可能模拟人类专家解决某些实际问题的决策和工作过程，即模仿人类专家如何运用他们的知识和经验来解决所面临问题的方法、技巧和步骤。我们这里就是将我们解题的技巧运用在编程之中。

2 《纸牌问题》选 自 IOI'99 第二试第二题，原题参见《信息学奥林匹克 》1999 年第 4 期。输入、输出同原题。

3《电阻问题》选自《电脑爱好者 》1999 年第 12 期，吴文虎等老师合办的擂台赛栏目。

4 《三角形灯塔》选 自 NOI'96 中的第二题。

这道题的方程解法简要叙述如下：

任一个灯的状态都可由它下方两个灯的状态求出。所以对于某一个已知状态的灯 b_{ij}(第 i 行第 j 个灯)，以该灯为顶点，两边向下延伸，与底边的 $x_j \cdots x_{n+j-i}$ 构成一个具有 N+1-i 行的的三角形。B_{ij} 的状态可由 $x_j \cdots x_{N+j-i}$ 推出，故可建立一个方程，有 p 个已知灯，则有 p 个一次线性方程，解此不定方程，问题得解。

5 《最优分解方案》选 自 IOI'96 中国队组队选拔赛。输入、输出同原题。

6 《排序网络问题》选 自 NOI'99 北京集训队训练题。

7、0、1 理论是解决排序网络问题的一种方法。简要地说，排序中的大小关系只有三种：大于、小于或等于。于是，验证一个 n 行的排序网络是否是完全排序网络，可通过对所有的由 0、1 组成的 n 位数列进行验证，二者完全等价。

8 《取数游戏》选 自 IOI'96 中的一道题。输入、输出同原题。

*9、NOI'99 福建省选拔赛中的《幂函数递归系数问题》是一道非常优秀的构造法题目。由于《信息学奥林匹克》99 年第 3 期中已经有了详尽的解法和证明，在此就没有加以叙述。

【参考书目】

1 《信息学奥林匹克》第一卷，第二卷 。

2 《IOI'98 中国集训队优秀论文集》 。

3《青少年国际和国内信息学（计算机奥林匹克竞赛指导）》，吴文虎、 王建德，清华大学出版社，1997。

4《国际国内奥林匹克信息学（计算机 ）1996 年竞赛试题解析》，吴文虎、王建德，清华大学出版社，1997。

5《图论及其应用》，卢开澄，卢开明，清华大学出版社 ，1995（第二版）

6《人工智能及其应用》，蔡自兴，徐光佑，清华大学出版社 ，1996（第二版）

7 《BASIC 语言——结构化程序设计》，谭浩强，中国科学技术出版社，1990

【源程序】

1、 例 1 的源程序

```
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q-,R-,S+,T-,V+,X+}
{$M 65520,0,655360}
Program Flat;
Type Array_type=array[1..200] of longint;{数组类型}
     Way_typep=array[1..10000] of integer;
     Way_type=array[1..2] of ^Way_typep;{移动方式数组类型}
Var Poles,{纸牌状态}
    Answers:Array_type;{方程的解，即每列需移动的总牌数}
    FinalWay:Way_type;{最佳移动方式}
    N:integer;{列数}
    FinalWaynum,{最佳移动方式的步数}
    Average:longint;{平均每列纸牌数}
    time:longint;{时间}
Procedure Init;{初始化}
var Total:longint;{纸牌总数}
    i:integer;
    filename:string;{输入文件名}
    f:text;{文件}
Begin {Init}
  time:=meml[$40:$6c];
  filename:='Flat.inp';
  assign(f,filename);
  reset(f);
  read(f,N);
  Total:=0;
  for i:=1 to N do
    begin
      read(f,Poles[i]);
      Total:=Total+Poles[i];{纸牌总数}
    end;
  Average:=Total div N;{最终状态 每列纸牌数}
  close(f);
End; {Init}

Procedure Compute;{解方程}
var Equations:Array_type;{方程}
    min:longint;{方程的最小的解}
    i:integer;
Begin {Compute}
  Equations[1]:=Poles[1]-Average;
  for i:=2 to N-1 do
    Equations[i]:=Poles[i]-Average;
  for i:=2 to N-1 do
    Equations[i]:=Equations[i]+Equations[i-1];
  Answers[1]:=0;
  for i:=2 to N do
    Answers[i]:=(Equations[i-1]-Answers[i-1])*(-1);
```

```
  min:=Answers[1];
  for i:=2 to N do
    if min>Answers[i] then min:=Answers[i];
  for i:=1 to N do
    Answers[i]:=Answers[i]-min;
End; {Compute}

Procedure Find;
Var Move,{纸牌状态}
    Answers_Find:Array_type;{每列所需移动总牌数}
    Choice:array[1..200,1..2] of Longint;{每列可移动的牌数}
    Way:Way_type;{移动方式}
    Waynum,{移动次数}
    Agreenum,{允许移动牌数}
    Movenum,{总剩余移动牌数}
    Change:longint;{交换用临时变量}
    i,j,k:integer;
Begin {Find}
  randomize;
  FinalWaynum:=1000000000;
  new(FinalWay[1]); new(FinalWay[2]);
  new(Way[1]); new(Way[2]);
  repeat {repeat}
    Movenum:=0;
    for i:=1 to N do
      Movenum:=Movenum+Answers[i];
    Move:=Poles; Answers_Find:=Answers;
    Waynum:=0;
    repeat{计算每列可移动的牌数}
      if Move[1]>=Answers_Find[1] then Choice[1,1]:=Answers_Find[1]
                                 else Choice[1,1]:=Move[1];
      if Move[N]>=Answers_Find[N] then Choice[N,1]:=Answers_Find[N]
                                 else Choice[N,1]:=Move[N];
      for i:=2 to N-1 do
        begin
          if Move[i]>=(Answers_Find[i] shl 1) then Choice[i,1]:=Answers_Find[i]
                                              else Choice[i,1]:=Move[i] shr 1;
          Choice[i,2]:=i;
        end;
      Choice[1,2]:=1; Choice[N,2]:=N;
      for i:=1 to 3 do{找出可移动的牌最多的三列}
        begin
          k:=i;
          for j:=i+1 to N do
            if Choice[k,1]<Choice[j,1] then k:=j;
          Change:=Choice[k,1]; Choice[k,1]:=Choice[i,1]; Choice[i,1]:=Change;
          Change:=Choice[k,2]; Choice[k,2]:=Choice[i,2]; Choice[i,2]:=Change;
        end;
      k:=random(3)+1;{在最多的三列中任取一列}
      {尽可能多地移动这一列的牌}
      if (Choice[k,2]>1) and (Choice[k,2]<N) then
```

```
        begin
          Agreenum:=Answers_Find[Choice[k,2]];
            if (Agreenum*2)>Move[Choice[k,2]] then Agreenum:=Move[Choice[k,2]]
div 2;
          Move[Choice[k,2]]:=Move[Choice[k,2]]-Agreenum*2;
          Move[Choice[k,2]+1]:=Move[Choice[k,2]+1]+Agreenum;
          Move[Choice[k,2]-1]:=Move[Choice[k,2]-1]+Agreenum;
          Answers_Find[Choice[k,2]]:=Answers_Find[Choice[k,2]]-Agreenum;
          Movenum:=Movenum-Agreenum;
        end;
      if (Choice[k,2]=1) or (Choice[k,2]=N) then
        begin
          Agreenum:=Answers_Find[Choice[k,2]];
          if Agreenum>Move[Choice[k,2]] then Agreenum:=Move[Choice[k,2]];
          Move[Choice[k,2]]:=Move[Choice[k,2]]-Agreenum;
          if Choice[k,2]=1 then
            Move[Choice[k,2]+1]:=Move[Choice[k,2]+1]+Agreenum
                          Else
            Move[Choice[k,2]-1]:=Move[Choice[k,2]-1]+Agreenum;
          Answers_Find[Choice[k,2]]:=Answers_Find[Choice[k,2]]-Agreenum;
          Movenum:=Movenum-Agreenum;
        end;
        if Agreenum>0 then
          begin
            inc(Waynum);
            Way[1]^[Waynum]:=Choice[k,2];
            Way[2]^[Waynum]:=Agreenum;
          end;
    until Movenum=0;{剩余的总移动牌数为零 即移动完成}
    if Waynum<FinalWaynum then begin FinalWaynum:=Waynum; FinalWay:=Way; end;
    {找到更优的移动方式则记录下来}
  until ((meml[$40:$6c]-time)/18.2)>2; {repeat}{时间允许的情况下不断重复}
End; {Find}

Procedure Print;{输出}
var i:integer;
    filename:string;{输出文件名}
    f:text;{文件}
Begin {Print}
  filename:='flat.out';
  assign(f,filename);
  rewrite(f);
  writeln(f,FinalWaynum);{输出总的移动次数}
  for i:=1 to FinalWaynum do{依次输出移动的方式}
    writeln(f,FinalWay[1]^[i]:3,'   ',FinalWay[2]^[i]:4);
  close(f);
End; {Print}

BEGIN {main}
  Init;
  Compute;
```

```
  Find;
  Print;
END. {main}
```

2、 例 2 的源程序

```
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q-,R-,S+,T-,V+,X+}
{$M 65520,0,655360}
PROGRAM Uncoil_Resistance;
Type array_type=array[0..100] of real;
Var Resistance:array[0..100] of ^array_type;{所有电阻}
    Equations:array[0..100] of ^array_type;{方程组}
    Num:array[1..100,1..100] of byte;{电阻编号}
    Line:array[1..100,1..2] of byte;{每个编号对应的电阻}
    Direct:array[1..100] of shortint;{路径方向}
    Mark:array[1..100] of shortint;{标记路径上结点的顺序}
    Connects:array[1..100,1..100] of shortint;{连通性}
    Points:set of 1..100;{路径中包含的点}
    Lines:set of 1..100;{路径中包含的电阻}
    N,
    M,
    l,
    a,b:integer;
    R:real;{总电阻}

Procedure Init;{读入电阻信息并初始化}
var filename:string;
    f:text;
    i,j,k,
    p,q:integer;
    js:integer;
Begin {Init}
  filename:='Input.txt';
  assign(f,filename);
  reset(f);
  readln(f,N);
  readln(f,M);
  readln(f,a,b);
  for p:=1 to 100 do
    begin
      new(Resistance[p]);
      new(Equations[p]);
      for q:=0 to 100 do
        begin
          Resistance[p]^[q]:=0;
          Equations[p]^[q]:=0;
        end;
    end;
  js:=0;
  for p:=1 to M do
    begin
```

```
      readln(f,i,j,k);
      Line[p,1]:=i; Line[p,2]:=j;
      if Resistance[i]^[j]=0 then begin
                                    Resistance[i]^[j]:=k;
                                    Resistance[j]^[i]:=k;
                                    Num[i,j]:=p; Num[j,i]:=p;
                                  end
                             else begin{将纯并联电阻合并}
                                       Resistance[i]^[j]:=1/(1/Resistance[i]^[j]
+1/k);
                                    Resistance[j]^[i]:=Resistance[i]^[j];
                                    Line[p,1]:=0; Line[p,2]:=0;
                                    inc(js);
                                  end;
    end;
  M:=M-js;
  close(f);
End; {Init}

Procedure Research_Circle;{寻找环}
var Way:array[1..100] of integer;
    Done,Wait:set of 1..100;
    escape:boolean;
    p,q,ll,i,j,k,x:integer;

Begin {Research_Circle}
  {构造路径}
  for i:=1 to N do
    Way[i]:=-1;
  Way[a]:=0; Done:=[]; Wait:=[a];
  repeat
    k:=10000; j:=-1;
    for i:=1 to N do
      if (i in Wait) and (k>Way[i]) then begin k:=Way[i]; j:=i; end;
    Done:=Done+[j]; Wait:=Wait-[j];
    for i:=1 to N do
      if (Resistance[j]^[i]>0) and ((Way[i]>(Way[j]+1)) or (Way[i]=-1)) then
        begin
          Way[i]:=Way[j]+1;
          Wait:=Wait+[i];
        end;
  until Wait=[];
  {记录路径上的点和路径方向}
  for i:=1 to M do
    Direct[i]:=0;
  i:=b; Points:=[b]; Lines:=[];
  repeat
    for j:=1 to N do
      if Way[j]=(Way[i]-1) then begin k:=j; j:=N; end;
    Points:=Points+[k];
    if k<i then Direct[Num[k,i]]:=1
```

```
          else Direct[Num[k,i]]:=-1;
    Lines:=Lines+[Num[k,i]];
    i:=k;
  until i=a;
  {扩充路径}
  repeat
    escape:=true;
    for i:=1 to M do
      if Direct[i]<>0 then
        for j:=1 to N do
                if (Not (j in Points)) and (Resistance[Line[i,1]]^[j]>0) and
(Resistance[Line[i,2]]^[j]>0) then
            begin
              escape:=false;
              Points:=Points+[j];
              Lines:=Lines+[Num[Line[i,1],j]];
              Lines:=Lines+[Num[Line[i,2],j]];
              Lines:=Lines-[Num[Line[i,1],Line[i,2]]];
              if Line[i,1]<j then Direct[Num[Line[i,1],j]]:=1
                             else Direct[Num[Line[i,1],j]]:=-1;
              if j<Line[i,2] then Direct[Num[Line[i,2],j]]:=1
                             else Direct[Num[Line[i,2],j]]:=-1;
              if (Line[i,1]<Line[i,2]) and (Direct[i]=-1) then
                begin
                  Direct[Num[Line[i,1],j]]:=-Direct[Num[Line[i,1],j]];
                  Direct[Num[Line[i,2],j]]:=-Direct[Num[Line[i,2],j]];
                end;
            end;
  until escape;
  {标记路径上结点的顺序}
  x:=1; Mark[a]:=x;
  q:=a; ll:=q;
  repeat
    for p:=1 to N do
      if (p<>q) and (Num[q,p] in Lines) and
                    (((p<q)  and  (Direct[Num[q,p]]=-1))  or  ((p>q)  and
(Direct[Num[q,p]]=1))) then
        begin ll:=p; p:=N; end;
    q:=ll;
    inc(x); Mark[q]:=x;
  until q=b;
End; {Research_Circle}

Procedure Connect;{路径之外其它点的连通性}
var i,j,k:integer;
Begin {Connect}
  for i:=1 to N-1 do
    for j:=i+1 to N do
      if (i<>j) and (Resistance[i]^[j]>0) and (not (Num[i,j] in lines)) then
        begin Connects[i,j]:=1; Connects[j,i]:=1; end
        else begin Connects[i,j]:=-9; Connects[j,i]:=-9; end;
```

```
  for i:=1 to N do
    Connects[i,i]:=0;
  for k:=1 to N do
    if not (k in points) then
    for i:=1 to N do
      if (i<>k) and (Connects[i,k]>0) then
        for j:=1 to N do
          if (i<>j) and (j<>k) and (Connects[k,j]>0) and
                              ((Connects[i,j]>(Connects[i,k]+Connects[k,j]))  or
(Connects[i,j]=-9)) then
            Connects[i,j]:=Connects[i,k]+Connects[k,j];
End; {Connect}

Procedure List;{列方程}
var i,j,k,p,q,ll:integer;
    Ways:array[1..100] of byte;
    pd:boolean;
Begin {List}
  for i:=1 to N do{对于每个结点}
    begin
      for j:=1 to N do
        if (i<>j) and (Resistance[i]^[j]>0) then
          begin
            if i<j then Equations[i]^[Num[i,j]]:=1
                   else Equations[i]^[Num[i,j]]:=-1;
          end;
      if i=a then Equations[i]^[0]:=-1;
      if i=b then Equations[i]^[0]:=1;
    end;
  Research_Circle;
  Connect;
  l:=0;
  for i:=1 to N do{对于环构造方程}
    for j:=1 to N do
       if (i<>j) and (i in Points) and (j in Points) and (Connects[i,j]>0) and
(Mark[i]<Mark[j]) then
        begin
          k:=2; Ways[1]:=i; Ways[2]:=0;
          repeat{查找环}
            pd:=false;
            for p:=Ways[k]+1 to N do
                              if  (Connects[Ways[k-1],j]=(Connects[p,j]+1))  and
(Resistance[Ways[k-1]]^[p]>0) then
                begin
                  Ways[k]:=p; Ways[k+1]:=0; k:=k+1; p:=N;
                  pd:=true;
                end;
            if not pd then k:=k-1;
            if Ways[k-1]=j then begin{找到了环}
                                l:=l+1;{添加一个方程}
                                for p:=1 to k-2 do
```

```
                                    begin
                                                    if Ways[p]>Ways[p+1] then
Equations[N+1]^[Num[Ways[p],Ways[p+1]]]:=1
                                                                        else
Equations[N+1]^[Num[Ways[p],Ways[p+1]]]:=-1;
                                    end;
                                  q:=i; ll:=q;
                                  repeat
                                    for p:=1 to N do
                                      if (p<>q) and (Num[q,p] in Lines) and
                                          (((p<q) and (Direct[Num[q,p]]=-1)) or
((p>q) and (Direct[Num[q,p]]=1))) then
                                            begin ll:=p; p:=N; end;
                                    Equations[N+1]^[Num[q,ll]]:=Direct[Num[q,ll
]];
                                    q:=ll;
                                  until q=j;
                                  for p:=1 to M do
                                    if Equations[N+1]^[p]<>0 then
                                      Equations[N+1]^[p]:=Equations[N+1]^[p]*Re
sistance[Line[p,1]]^[Line[p,2]];
                                end;
          until k=1;
        end;
  inc(l);
  for p:=0 to M do
    Equations[N+1]^[p]:=0;
  j:=a; k:=j;
  repeat
    for p:=1 to N do
      if (p<>j) and (Num[j,p] in Lines) and
                        (((p<j)   and   (Direct[Num[j,p]]=-1))   or   ((p>j)   and
(Direct[Num[j,p]]=1))) then
    begin k:=p; p:=N; end;
    Equations[N+1]^[Num[j,k]]:=Direct[Num[j,k]];
    j:=k;
  until j=b;
  for p:=1 to M do
                                   if         Equations[N+1]^[p]<>0         then
Equations[N+1]^[p]:=Equations[N+1]^[p]*Resistance[Line[p,1]]^[Line[p,2]];
End; {List}

Procedure Uncoil;{利用高斯消元法，解方程组}
Var Answers:array[0..100] of real;{方程组的解}
    i,j,k:integer;
    U:integer;
    Change:real;
    Compute:real;
Begin {Uncoil}
  for i:=1 to M+1 do
    begin
```

```
      if Equations[i]^[i-1]=0 then
        begin
          for j:=i+1 to N+1 do
            if Equations[j]^[i-1]<>0 then begin k:=j; j:=N+1; end;
          for j:=i-1 to M do
            begin
              Change:=Equations[i]^[j];
              Equations[i]^[j]:=Equations[k]^[j];
              Equations[k]^[j]:=Change;
            end;
        end;
      for j:=i+1 to N+1 do
        if Equations[j]^[i-1]<>0 then
          begin
            for k:=i to M do
              Equations[j]^[k]:=Equations[i]^[k]*Equations[j]^[i-1]/Equations[i
]^[i-1]-Equations[j]^[k];
            Equations[j]^[i-1]:=0;
          end;
    end;
  U:=100;{代入求解}
  Answers[M+1]:=U/Equations[M+1]^[M];
  for i:=M downto 1 do
    begin
      Compute:=0;
      for j:=M downto i do
        Compute:=Compute+Equations[i]^[j]*Answers[j+1];
      Answers[i]:=-Compute/Equations[i]^[i-1];
    end;
  R:=U/Answers[1];{计算总电阻}
End; {Uncoil}

Procedure Print;{输出总电阻}
Begin {Print}
  writeln(abs(R):0:2);
End; {Print}

BEGIN {main}
  Init;
  List;
  Uncoil;
  Print;
END. {main}
```

3、 例3的源程序

```
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q-,R-,S+,T-,V+,X+}
{$M 65520,0,655360}
Program MaxMul;
Const Max=1000;{N的最大值}
Type Multype=array[0..255] of integer;{存储乘积的类型}
```

```
Var D:array[1..50] of integer;{存储分解的数}
    i,j,n,m:integer;
    Mul:MulType;{乘积}
    Change:Boolean;

Procedure Multhem(n:integer);{Mul=Mul*N}
Begin
  for i:=1 to Mul[0] do
    Mul[i]:=Mul[i]*n;
  for i:=1 to Mul[0] do
    begin
      Mul[i+1]:=Mul[i+1]+Mul[i] div 10;
      Mul[i]:=Mul[i] mod 10;
    end;
  i:=Mul[0]+1;
  While Mul[i]<>0 do
    begin
      inc(Mul[0]);
      Mul[i+1]:=Mul[i+1]+Mul[i] div 10;
      Mul[i]:=Mul[i] mod 10;
      inc(i);
    end;
End;

Procedure GetMul;{将分解出来的数乘起来放在Mul中}
Var i,j:byte;
Begin
  fillchar(Mul,sizeof(Mul),0);
  Mul[0]:=1;
  Mul[1]:=1;
  for i:=1 to m do
    Multhem(D[i]);
End;

BEGIN {main}
  repeat
    write('N=');
    readln(n);
    if (n>Max) or (n<3) then writeln('Out of Range!');
  until (n<=Max) and (n>=3);
  j:=2;
  m:=1;
  While (n>=2*j+1) do{将n拆成尽可能多的不同的大于1的数}
    begin
      D[m]:=j;
      n:=n-j;
      j:=j+1;
      m:=m+1;
    end;
  D[m]:=n;
  for i:=m-1 downto 1 do{调整这些数，使它们尽量“紧密”}
```

```
    if ((i=m-1) and (D[m]>D[m-1]+2)) or
       ((i<>m-1) and (D[m]>D[m-1]+1)) then
      begin
        dec(D[m]);
        inc(D[i]);
      end
      else Break;
  for i:=1 to m-1 do{输出这些数}
    write(D[i],',');
  writeln(D[m]);
  GetMul;{求它们的乘积}
  For i:=Mul[0] downto 1 do{输出乘积}
    write(Mul[i]);
  writeln;
END. {main}
```

4、 例 4 的源程序

```
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q-,R-,S+,T-,V+,X+}
{$M 65520,0,655360}
Program Taxis_Net;
Var Inequality:array[1..100,1..100] of shortint;{存储不等式组}
    N,M:word;{N 导线条数; M 电桥数}
Procedure Work;
Var p,
    a,b,c,
    Change:integer;
Begin {Work}
  assign(input,'Input.txt');
  reset(input);
  readln(N);
  readln(M);
  fillchar(Inequality,sizeof(Inequality),0);
  for p:=1 to M do{依次处理每一个电桥}
    begin
      readln(a,b);
      if a>b then begin
                    Change:=a;
                    a:=b;
                    b:=Change;
                  end;
      if Inequality[a,b]=0 then
        begin
          Inequality[a,b]:=1; Inequality[b,a]:=-1;
          for c:=1 to N do
            if (c<>a) and (c<>b) and (Inequality[a,c]<>0) then
                  {调整所有关于 a 的不等式}
              begin
                if (Inequality[c,a]=1) and (Inequality[c,b]=0) then
                  begin
                    Inequality[c,b]:=1; Inequality[b,c]:=-1;
```

```
                  Inequality[c,a]:=0; Inequality[a,c]:=0;
                end;
              if (Inequality[c,a]=-1) and (Inequality[c,b]=0) then
                begin
                  Inequality[c,a]:=-1; Inequality[a,c]:=1;
                  Inequality[c,b]:=0; Inequality[b,c]:=0;
                end;
            end;
          for c:=1 to N do
            if (c<>a) and (c<>b) and (Inequality[c,b]<>0) then
                  {调整所有关于 b 的不等式}
              begin
                if (Inequality[c,b]=1) and (Inequality[c,a]=0) then
                  begin
                    Inequality[c,b]:=1; Inequality[b,c]:=-1;
                    Inequality[c,a]:=0; Inequality[a,c]:=0;
                  end;
                if (Inequality[c,b]=-1) and (Inequality[c,a]=0) then
                  begin
                    Inequality[a,c]:=1; Inequality[c,a]:=-1;
                    Inequality[c,b]:=0; Inequality[b,c]:=0;
                  end;
              end;
        end;
    end;
  close(input);
  assign(output,'Output.txt');
  rewrite(output);
  for p:=1 to N-1 do
    if Inequality[p,p+1]<>1 then begin{判断排序网络是否符合条件}
                                   writeln('No!');{不符合条件}
                                   close(output);
                                   halt;
                                 end;
  writeln('Yes!');{符合条件}
  close(output);
End; {Work}

BEGIN {main}
  Work;
END. {main}
```

5、 例 5 的源程序

```
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q-,R-,S+,T-,V+,X+}
{$M 65520,0,655360}
Program IOI96_Program3;
Uses Play;
Const Max=100;{N 的最大值}
Var n:byte;
    D:array[1..Max] of byte;{排列中的 N 个数}
```

```
    MyScore,YourScore,
    Start:integer;
    SelectOdd:boolean;{选择奇数位还是偶数位}

Procedure Init;{读文件和初始化过程}
Var i,j:integer;
    Odd,Even:integer;{Odd 奇数位的和;Even 偶数位的和}
Begin
  assign(input,'Input.txt');
  reset(input);
  readln(n);
  for i:=1 to n do
    readln(D[i]);
  close(input);
  Odd:=0;
  Even:=0;
  for i:=1 to n div 2 do{求出奇数位和偶数位的和}
    begin
      Odd:=Odd+D[i*2-1];
      Even:=Even+D[i*2];
    end;
  if Odd>Even then begin{如果奇数位的和大}
                     MyScore:=Odd;{选择拿所有奇数位的数字}
                     YourScore:=Even;
                     SelectOdd:=true;
                   end
              else begin{否则}
                     MyScore:=Even;{选择拿所有偶数位的数字}
                     YourScore:=Odd;
                     SelectOdd:=false;
                   end;
End;

Procedure Task;
Var i:integer;
    C:Char;
Begin
  StartGame;
  Start:=1;
  for i:=1 to n div 2 do
    begin
      if Odd(Start)=SelectOdd then begin
                                     MyMove('L');
                                     inc(Start);
                                   end
                              else begin
                                     MyMove('R');
                                     YourMove(C);
                                     if C='L' then inc(Start);
                                   end;
    end;
```

```
  assign(output,'Output.txt');
  rewrite(output);
  writeln(MyScore,' ',YourScore);
  close(output);
End;

BEGIN {main}
  Init;
  Task;
END. {main}
```

附上例 5 中用于测试的 PLAY 单元。

```
UNIT PLAY;

INTERFACE
  PROCEDURE STARTGAME;
  PROCEDURE MYMOVE(C:CHAR);
  PROCEDURE YOURMOVE(VAR C:CHAR);

IMPLEMENTATION

PROCEDURE STARTGAME;
Begin

End;

PROCEDURE MYMOVE(C:CHAR);
Begin

End;

PROCEDURE YOURMOVE(VAR C:CHAR);
Begin
  C:='L';
End;

END.
```