规模化问题的解题策略

湖南省长沙市第一中学　谢婧

【关键字】 规模化　策略　算法

【摘要】

问题规模化是近来信息学竞赛的一个新趋势，它意在通过扩大数据量来增加算法设计和编程实现的难度，这就向信息学竞赛的选手提出了更高层次的要求，本文试图探索一些解决此类问题的普遍性的策略。开始，本文给出了“规模化”一词的定义，并据此将其分为横向扩展和纵向扩展两种类型，分别进行论述。在探讨横向扩展问题的解决时本文是以谋划策略的“降维”思想为主要对象的；而重点讨论的是纵向扩展问题的解决，先提出了两种策略——分解法和精简法，然后结合一个具体例子研究“剪枝”在规模化问题中的应用。问题规模化是信息学竞赛向实际运用靠拢的一个体现，因此具有不可忽视的意义。

【正文】

一　引　论

（一）背景分析

分析近年来国际、国内中学生信息学竞赛试题，可以看出信息学竞赛对于选手的要求已经不再仅仅局限于“算法设计”，它同时在编程实现方面加强了考察力度，由侧重于考察理论知识转向理论考察与实践考察并重。这一命题宗旨的转变，给信息学竞赛注入了新的机能，为命题者开拓了另一个领域。其一体现有：试题由精巧型(这类试题的难度主要体现在精妙算法的构造，属于一点即通的类型)向规模型发展，从而使得问题的实现复杂化。

（二）对“规模化”的理解

规模一词在字典中的含义是：事物所具有的格式、形式或范围。在信息学竞赛中，问题的规模具体是指待处理数据量的大小，通常可以通过一组规模参数(S_1,S_2,…, S_k)来表示。例如下列问题 1 的规模就是(100)，而问题 2 的规模是(100,100)。

问题 1：求数列的前 100 项之和。

问题 2：求 100*100 的矩阵中的各项之和。

问题 3：求数列的前 1000 项之和。

“规模化”即扩展问题的规模，它具体是指增加规模参数的个数或扩大规模参数的数值范围。我们知道，如果撇开计算机的硬件、软件等环境因素，可

以认为一个特定算法的“运行工作量”的大小，只依赖于问题的规模，或者说，它是问题规模的函数，程序的执行时间与存储量需求直接受到问题规模的影响。由于种种现行条件的制约，随着规模扩展，问题的实际解法集便会缩小，甚至变为空集，这有时会使问题规模扩展后无法用原来小规模时的理想模型解决。如 NOI’99《生日蛋糕》一题，理论上可以用动态规划的方法求解，但因其空间耗费过大，多数人是用搜索来实现的。

从“规模化”一词的定义不难看出，它包括横向扩展和纵向扩展。横向扩展是指增加规模参数的个数，如由问题 1 扩展至问题 2，即我们通常说的多维化；纵向扩展是指扩大规模参数的数值范围，如由问题 1 扩展至问题 3。下文将分别探讨这两类问题的一般性解题策略。

二 横向扩展问题的解题策略

（一）构造策略的思想

横向扩展问题一般具有维数高、难于构想的特点，所以谋划解决这一类问题的策略，通常采用“降维”[1]的思想：分析低维问题，找到解法，推广至高维的情况。

下面我们就来看一个具体例子。

问题一：对于一个 n 维体 P((S1,T1),(S2,T2),...,(Sn,Tn))，Si、Ti (i=1..n)均为整数，我们定义其阶积=(T1-S1) *(T2-S2)*...*(Tn-Sn)，并称(Ti-Si)是 P 的一个要素 i。

如果存在另一个 n 维体 Q((S1`,T1`),(S2`,T2`),...,(Sn`,Tn`))，使得 Si`≥Si(i=1..n)，且 Ti`≤Ti(i=1..n)，Si`、Ti` (i=1..n)也是整数，则称 Q 是 P 的子 n 维体。

现给定一个 n 维体((0,A1),(0,A2)，...，(0,An))，求它所有子 n 维体的阶积和。

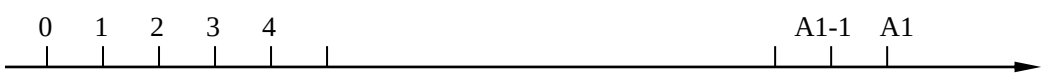
〖问题分析〗:

如果泛泛地从 n 维体入手，会觉得无所适从，根据要求“所有子 n 维体的阶积和”，我们可以枚举所有的子 n 维体，其时间复杂度高达 O(m^{2n})（其中 m 表示 Ai(i=1..n)的一般规模），效率不高的主要原因是数学模型不够抽象，而好的数学模型是建立在问题本质基础上的。所以说，如果我们对问题缺乏认识或认识不深，就不可能高效地解决它。这就是笼统的考虑横向扩展问题的弊病。

下面我们根据上文提到的“降维”思想来解决此题。

第一步：降低问题的规模。

我们先从简单模型入手，来看一看 n=1 时的情况，我们把一维体((0，A1))体现在在下图所示的一根数轴上，这里不妨把一维体看成一条线段，其阶积就是线段的长度。



第二步：在低维问题中求找规律。

试想把长度相同的子线段归类统计，那么对于长度为 L 的线段(s,s+L)：

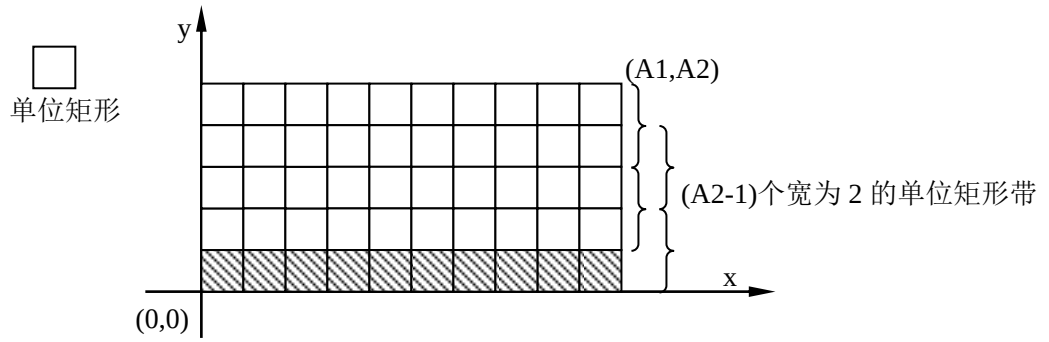
∵s+L≤A1 ∴ s≤A1-L 又∵s≥0，

∴0≤s≤A1-L ，这样的子线段共有 A1+1-L 条。

所以，一维体（(0,A1)）的所有子一维体的阶积和为Σi*(A1+1-i){i=1..A1}，设为 Fg(A1)。

第三步：将规律推广至高维问题。

我们将模型稍加推广，看看 n=2 时的情况。这时我们可将二维体看成一个矩形，其阶积就是矩形的面积。



在上图中，我们把一个矩形嵌入平面直角坐标系。这里我们按照子矩形不同的长(x 轴上的距离)、宽(y 轴上的距离)来统计。

我们先提取矩形中一个宽为 1 的单位矩形带(如上图的阴影部分)，然后讨论矩形的长。根据解决一维体时的规律，我们知道在这个单位矩形带中长为 L 的矩形共有 A1+1-L 个，所以在单位矩形带中，所有子矩形的面积和为 Fg(A1)。由于宽为 1 的单位矩形带在原矩形中共有 A2 个，所有宽为 1 的子矩形的面积之和为 1*A2* Fg(A1)。

同理，所有宽为 2 的子矩形的面积之和为 2*(A2-1)*Fg(A1)，因此所有宽为 W 的子矩形的面积之和为 W *(A2+1-W)* Fg(A1)。由此可知二维体所有子二维体的阶积之和是 Fg(A2)* Fg(A1)。

逐步推广，可以得知求 n 维体（(0,A1),(0,A2)，…，(0,An)）所有子 n 维体的阶积和为 Fg(A1)* Fg(A2)*…* Fg(An)。其中，

$$Fg(a)=(1+2+\ldots+a)+(1+2+\ldots+a-1)+\ldots+(1)$$

$$=\frac{1}{2}[(a+1)a+a(a-1)+(a-1)(a-2)+\ldots+2]$$

$$=\frac{1}{6}[(a+2)(a+1)a-(a+1)a(a-1)+(a+1)a(a-1)-a(a-1)(a-2)+\ldots+6-0]$$

$$=\frac{1}{6}a(a+1)(a+2)$$

至此，问题得到圆满解决，时间复杂度已经降到 0(n)，足够满足维数高的

情况。

（二）小　结

当然了，大多数横向扩展问题最终并不能如此轻松地解决，实际竞赛中的问题是非常复杂的，上面列举的例子没有涉及其他方面的知识点，是为了集中说明具体如何运用“降维”思想来分析问题。横向扩展问题的难度主要体现在思维上，所以我们应当从低维的简单情况入手，通过挖掘低维问题与高维问题的相通之处来寻找规律，找到规律后不能机械地推广到高维模型，要注意灵活、变通，真正使它发挥作用。

三　纵向扩展问题的解题策略

（一）分解法

问题二：求正整数 N 和 M 之间具有最多真因子的数。本题中的真因子是这样定义的：如果 R<P 而且 R 能整除 P，我们就称 R 是 P 的真因子，对于特殊整数 1，我们认为 1 是 1 的真因子。

参数范围：1≤N＜M≤999999999；M-N＜999999；

时限：10s。

我们很容易得到下列两个方法：

<方法一>顺序查找法：依次统计规定范围内的各整数的真因子个数，记录最优解。

由于，分解质因数的算法时间复杂度为平方根级的，因此这个算法的时间复杂度为 $0((m-n)*m^{0.5})$。

<方法二>标号法：枚举不同的因数，标记它们的倍数。

如果不仔细分析，会认为两种方法的算法时间复杂度一样，实际上后者的时间复杂度是 $0((m-n)*(1+1/2+1/3...+1/[m^{0.5}]))$，还不到 $0((m-n)*[\log_2 m^{0.5}])$ {[x]表示[x,x+1)间的整数}。证明如下：

先用数学归纳法证明 $1+1/2+1/3+1/4+1/5+\ldots+1/(2^n-1)\leqslant n$。

当 n=1 时，左边=1，右边=n=1；1≤1，不等式成立。

假设当 n=k 时，不等式成立，则有

$1+1/2+1/3+1/4+1/5+\ldots+1/2^k-1\leqslant k$

现证明 n=k+1 时，不等式依然成立，

∵ $1/2^k+1/(2^k+1)+1/(2^k+2)+\ldots+1/(2^{k+1}-1) < 1/2^k+1/2^k+\ldots+1/2^k$

$=(2^{k+1}-1-2^k+1)/2^k$

$=1$

∴ $1+1/2+1/3+\ldots+1/2^k-1+1/2^k+1/(2^k+1)+\ldots+1/(2^{k+1}-1)\leqslant k+1$

即 $1+1/2+1/3+1/4+1/5+\ldots+1/2^{k+1}-1\leqslant k+1$

故命题成立。

所以，$1+1/2+1/3+1/4+1/5+\ldots+1/n\leqslant[\log_2 n]$

方法二之所以在时间复杂度上大有降低，是因为它采用了“空间换时间”

的模式，由于在标号的全过程中必须保存当前各整数的真因子个数，因此空间复杂度是 O(m-n)，从参数范围可知，实际情况下无法满足这一需求。它仅仅停留在理论基础上，无法用程序实现。方法一虽然空间耗费小，具有可行性，但时间耗费却难以满足要求。于是我们得到：

<方法三>分段统计法：将给定区间分成不重复且不遗漏的若干个子区间，然后按方法二统计。

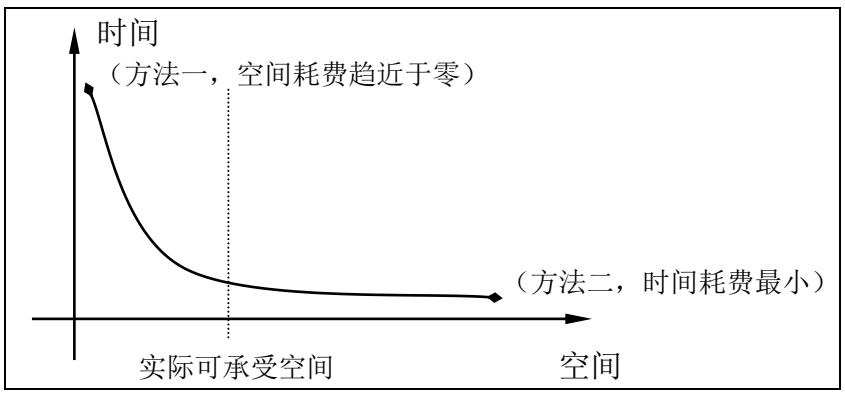
由于方法一每次处理单一元素，因此时间耗费高，方法二将所有元素统一处理，因此空间需求大，而方法三则综合前两种方法的优点，在充分利用空间的情况下，得到较高的时间效率。

方法三实质就是分解法的应用，由此我们将“分解法”定义如下：

以一定的算法为原型，将大规模的问题分解成若干个不遗漏且尽量不重复的相对独立的子问题，使得所有子问题解集的全集就是原问题的解集。

分解法的原理和适用范围：

解决某些纵向扩展问题的时候，常常会出现理论需求与实际承受能力之间的“矛盾”，它主要体现在时空需求互相制约的关系上。如本题中的时空关系可以用下图所示的曲线（双曲线的某一支的一部分）来表示，其中曲线的两个端点分别代表方法一与方法二的时空需求。这时若把问题分解成若干规模较小的子问题，套用原有的算法解决，就能有效地中和时空需求的矛盾。通常，我们以实际空间承受能力作为划分子问题的规模标准，这样才能令时间效率得到最大提高。下图中，虚线位置表示实际空间承受能力的上限，它与曲线的交点就是时空需求分配的最优方案。



（二）精简法

我们对“精简法”定义如下：

忽略问题的表面因素，只提取具有实质性联系的特殊信息，以节省空间适应问题的规模。

下面我们结合一个具体例子说明这一解题方法：

问题三：最长词链问题。

给定一个仅包含小写字母的英文单词表，其中的每个单词包含至少 1 个字母，至多 75 个字母，且按字典顺序由小到大排列（不会重复）。所有单词所含的字母个数总和不超过 2,000,000 个。

如果在一张由一个词或多个词组成的表中，每个单词（除最后一个）都被其后一个单词所包含（是其前缀），则称此表为一个链。

现要求从单词表中找到一个包含单词数最多的最长词链。

〖问题分析〗问题的实质是在一定的序列中，求找相邻元素（字串）间存在特定关系（包含）的最长子序列。解决这类问题，通常会用到动态规划的办法。这在求“数列的最长非降子序列”和 IOI’99《隐藏的码字》一题中都有所运用。

我们令 num(i)表示以第 i 个单词为链尾的最长词链，则

num(i)=max{num(j)}+1{j＜i，且单词 j 是单词 i 的前缀}

所以，空间复杂度为 O(n){n 为所包含的单词个数}，本题的数据量过大，显然无法满足这一存储量需求。

仔细考虑“字典顺序”和“前缀”的关系可以得出这样的结论：两个单词 w1、w2，如果 w2 包含 w1，且有一个单词 w3 满足，w1<w3，w3<w2，那么 w3 必定包含 w1。简证如下：

令 w2=$L_1L_2L_3..L_{LW2}$，w3=$L_1`L_2`L_3`..L_{LW3}`$，则 w1=$L_1L_2L_3..L_{LW1}$(L_i 或 $L_i`$为字母)。

假设 w3 不包含 w1，由于 w1<w3，则必存在一个 i 值(i≤Lw1，i≤Lw3)，使得 $L_j=L_j`$(0＜j＜i)，$L_i＜L_i`$，所以 w3>w2，这与 w3<w2 矛盾，因此假设不成立。

所以某个单词的(单词表中存在的)前缀，必定被其前驱单词所包含或者就是其前驱单词。于是我们可将单词的所有前缀(每个单词也可视为自身的前缀)标记出来，来为后面的单词传递信息。如下表：

i；int；integer；intern；internet；

i	n	t	e	g	e	r	

i	n	t	e	r	n		

i	n	t	e	r	n	e	t

首先，i 和 int 被标记为后继单词 integer 的前缀，待读入单词 intern 后，我们只需将单词 integer 与 intern 比较，就得出了它的前缀词链 i 和 int，最后处理 internet，得出上表中的最长词链为 i；int；intern；internet。

这样问题的空间复杂度就由 O(n)降低至常量阶 O(1)。下面我们再来分析两种方法的时间复杂度，如果以对两两单词间的比较作为基本运算的原操作，则动态规划方法的时间复杂度为 $O(n^2)$，后者的时间复杂度仅为 O(n)。我们发现，两种方法不存在时空需求上的制约关系，因此分解法便无“用武之地”了。动

态规划的方法尽管在求“数列的最长非降子序列”中得到了很好的效果，但却不适应本题，主要基于这几个原因：⑴字串的存储比数列元素复杂，使得空间耗费大；⑵字串的包含关系比数列元素的大小关系复杂，使得时间耗费大；⑶单词表的元素存在有序性，使得我们能够根据问题良好的特性设计最为适用的算法。方法二巧妙抓住了两两元素（单词）间的内在联系，提炼出了问题的关键，使得处理对象明确化。

精简法的适用范围：

处理对象的存储空间大，操作比较复杂(例如对字符串的操作)，这样的元素间往往存在一定的特殊关系，于是我们可以仅仅提取问题的脉络，而实际的元素能够附着于其上，这就将看似凌乱孤立的元素转化成了具有一定逻辑关系的结构。

其实，多数人对“精简法”并不陌生，大家在处理 01 串时，常常会把它转化为对应的十进制数，使得离散的元素构成了线性结构，这同样也是精简法的一大应用。

精简法的特点：

精简法在于抓住问题的本质，升华元素的内在联系，淡化元素的孤立性，将大规模的数据抽象成简单、明了的关系，使得问题更易于描述，实际操作更加简化。因此，精简法针对性强，设计算法时没有固定模式可套，需要具体问题具体对待。

（三）巧用剪枝

统计问题是纵向扩展问题的一大组成部分，由于这类问题大多可以采用时间复杂度为多项式阶 O（n^k）的算法解决，难度不大，所以长久以来难以在大赛中显露头角。近来，命题者通过扩大统计问题的规模来增加其难度，于是，统计问题便开始活跃于重大的信息学竞赛中，如 NOI’97 的《卫星覆盖》，IOI’98 的《图形周长》。

剪枝就是通过某种判断，筛减掉一些不必要的计算过程。它源于搜索算法，好的剪枝条件，往往能够极大提高搜索的时间效率。然而，由于多项式阶的算法通常被视为有效算法，因而很少有人问津剪枝条件，甚至认为那起不了多大作用。那么到底剪枝的应用对于统计问题有多大的作用呢？还是让我们先来看一个具体例子吧。

问题四：IOI’99《机场跑道》。

试题简述：在一个数字矩阵中求解一个最大数与最小数差不超过阈值 C 的面积最大的子矩阵。

<方法一>记录下每行中包括首元素在内的下限一定的最大线性区域长度，然后查找下限一定的面积最大区域。

(为方便讨论，在示意图中行、列的排列按从上至下，从左至右的顺序)

38	38	33	39
39	40	39	39
39	40	41	38
36	39	39	39

以左图的一个 4*4 的矩形为例（C=4）：

从（1,1）格开始，下限为 36（上限为 36+C）的线性区域可延伸至（1,2）格；

同理，从（2,1），（3,1），（4,1）格开始，下限为36的线性区域分别可延伸至(2,4)，(3,2)，(4,4）格。 所以下限为 36 的面积最大区域为（1,1,4,2）。

我们注意到，统计每行的最大线性区域时，由于必须包括首元素，因此下限值与首位元素值不会超过 C。由于 C 的取值范围是[0,10]，空间需求可以满足，该算法具有可行性，其算法复杂度为 $O(uv^2C)$。由于 u、v 的上限是 700，尽管试题规定的最大时限长达一分钟，仍然无法满足这一时间耗费。

<方法二>将矩阵进行横向压缩，得到一系列单位区域，然后求最长的连续单位区域。

我们可以得到如下算法：

1 确定西界限；
2　确定东界限（＜西界限+100）；
3　　对从西界限至东界限的每个单位区域进行统计；
4　　确定北界限
5　　找到最小的南界限；

可以看出，上述算法的时间复杂度是 $O(100uv^2)$。

如果从时间复杂度来看两种方法，前者肯定比后者好。但如果对于后者加上好的剪枝条件，结果就不一样了。

由于我们求找面积最大的矩形区域会不断更新当前的最优值，所以，如果某次计算所能得到的最大面积不超过当前的最优值，则这样的计算毫无意义，可以省略。

38	38	33	39
39	40	39	39
39	40	41	38
36	39	39	39

38	38	33	39
39	40	39	39
39	40	41	38
36	39	39	39

38	38	33	39
39	40	39	39
39	40	41	38
36	39	39	39

我们仍然以原来 4*4 的矩形为例，以上三个矩形中分别用粗线条框出了西界限为 1，东界限为 2、3、4 时的面积最大区域。我们可以发现南北界限的差别呈递减趋势，事实上，这并不是偶然的，可以用反证法证明，西界限一定，而东界限逐步扩展，也即矩形的宽度增加时，最大面积矩形区域的长度是（非严格）递减的。

我们令 Max_wide[wno,eno]表示东西界限为 eno 和 wno 时的最大区域长度，那么 Max_wide[wno,eno] ≤Max_wide[wno,eno-1]。所以我们可以得到下列两个

剪枝条件：

If　最大宽度（Min{100,u}）* Max_wide[wno,eno-1]<当前最大面积
　Then　扩展西界限；
If　(eno- wno+1)* Max_wide[wno,eno-1]<当前最大面积
　Then　扩展东界限；

下面我们就将方法一（land_1.pas）、方法二（land_2.pas）及结合剪枝条件后的方法二（land_3.pas）进行测试时间的对照：

输入数据	数据规模 (U,V,C)	运 行 时 间		
		Land_1.pas	Land_2.pas	Land_3.pas
1	20，20，0	0.05s	0.05s	<0.05s
2	30，41，5	0.05s	0.49s	<0.05s
3	100，100，2	0.22s	0.94s	0.11s
4	100，500，7	1.26s	2.20s	0.90s
5	300，300，10	9.23s	10.33s	2.47s
6	500，400，10	20.99s	98.52s	7.25s
7	600，500，5	1.54s	25.73s	4.89s
8	700，700，9	99.89s	56.87s	9.78s
9	700，700，10	168.85s	1695.93s	34.12s

（运行环境：pentium 166MHz/16MB）

注：数据是按规模从大到小进行排列的。

测试结果分析：

⑴除数据 7 的运行时间与数据规模不相称外，程序一的运行时间与数据规模相对稳定，那主要是由于数据 7 中相邻正方形的高度差大多大于阈值 C。所以 $O(uv^2C)$是本算法的平均时间复杂度，它能够较为准确地反映其时间耗费。

⑵对于规模递增的数据，程序二的运行时间波动很大，尤其是数据 8 与数据 9，两数据规模相差无几，而运行结果却大相径庭。这说明了 $O(100uv^2)$仅仅是算法最坏情况下的时间复杂度。由于程序二的实际时间耗费对于数据规模的依赖性大，因此难以用时间复杂度较准确地反映。

剪枝效果分析：

一旦确定剪枝条件，每次统计都会执行一次判断操作，所以不精准的剪枝会带来很大的负面影响，再则剪枝条件的效果常常被认为存在很大的偶然性。所以它常常被人们冷落。

在上表中，相对于程序二，程序三的运行速度大大提高，且普遍优于程序一，对比说明剪枝条件还是起到了很好的筛减作用。从其运行时间中，我们还能够大致看出数据规模，尽管我们并不能精准地估计出剪枝条件的效用到底有多大，但这一相对稳定性足以说明，使用优秀的剪枝条件或者综合使用多方面的剪枝条件，收益良好也就是“偶然”中的必然了。尽管剪枝不能降低算法的时间复杂度，但却对降低实际时间耗费有着非同小可的作用。尤其是规模化问题，时间耗费大，如果注意分析问题，找到约束信息，必将起到事半功倍的效果。

（四）小 结

由于纵向扩展问题具有很大的灵活性，很难像多维化问题那样总结出一个统一的谋划策略的思想，也难以归纳出较为完整的策略集，以上仅仅是我根据平时练习的经验总结出的自认为有一定推广意义的一些策略，其中也提到了剪枝在纵向扩展问题中的应用，尽管它不是一个具体的策略，但对于规模化问题也有普遍意义，这已经在上文中有所提及。总而言之，我们研究各种策略的目的是一致的，那就是要有效地解决问题。相信通过不断的学习，并与其他同学以及教练们进行交流探讨，一定会探索出更多，更具价值的策略。

四 结 语

我们知道，无论算法如何优化，所解决的问题规模终究是有限的，因此解决规模化问题时，首先应明确问题的实际规模，然后量“体”裁“衣”，设计适用的算法。有时题中还会对问题规模加上了一定的限制条件，例如 IOI’99《隐藏的码字》中，文本的最大长度为 1,000,000，而“右侧最小”覆盖序列的总数却不超过 10,000 个，这就使得问题的实际规模大大降低。所以，审好题是解决规模化问题的第一步。

我们在信息学竞赛中所解决的问题都是经过实际问题理想化后的产物，而在现实生活中，真正需要处理和解决的是很大规模的数据量，这就是我们现在涉足规模化问题的一大现实意义，同时规模化问题也向我们提出了更高层次的要求。值得注意的是，规模化问题固然重要，但它并不是空中楼阁，它是建立在小规模问题基础上的，因此，想要很好地解决规模化问题，必须从基础做起，这样的道理也不只是适应于信息学竞赛的。既具备扎实的理论基础，又具有实干本领和创新精神，是时代对于跨世纪青年的基本要求，也是我们不息奋斗的目标。

【附录】

[1] 本文提到的“降维”是一种谋划策略的思想，与通常意义上的降维一定区别。

【程序】

1 由于枚举法效率太低，因此仅给出有效算法的程序。

```
program object;
type
  node=record{存储高精度整数}
         v:array[1..100] of longint;
         last:integer;
       end;
var
  n,i:integer;
  a:array[1..10] of integer;{存储 n 维体的信息}
  tot:node;{阶积和}

procedure readn;{读入 n 维体的信息}
var i:integer;
    f:text;
begin
  assign(F,'input.txt');reset(F);
  readln(F,n);
  for i:=1 to n do read(f,a[i]);
  close(f);
end;

function num(s:integer):longint;
begin
  num:=(s+2)*(s+1)*s div 6;
end;

procedure multiply(var nd:node;s:longint);{高精度乘法}
var c:longint;
    i:integer;
begin
  c:=0;
  with nd do begin
    for i:=1 to last do
```

```
      begin
        v[i]:=v[i]*s+c;
        c:=v[i] div 10;
        v[i]:=v[i] mod 10;
      end;
    while c<>0 do begin
      inc(last);
      v[last]:=c mod 10;
      c:=c div 10;
    end;
  end;
end;

begin
  readn;
  with tot do
    begin
      fillchar(v,sizeof(v),0);
      v[1]:=1;last:=1;
    end;{初始化}
  for i:=1 to n do multiply(tot,num(a[i]));{计算阶积和}
  with tot do{输出结果}
    for i:=last downto 1 do write(v[i]);
  writeln;
end.
```

2 由于方法一效率太低，方法二无法实现，仅给出对应方法三的程序。

```
program number;
const inputfile='number.in';
      outputfile='number.out';
var
  f:text;
  n,m:longint;
  mb:longint;{储存当前真因子个数最多的数}
  fn:integer;{储存当前真因子个数最多的数所包含的真因子个数}
  s,t,i,j:longint;{ s+1,t 为分段统计时的起点、终点}
  a:array[1..30000] of integer;
 {每次统计连续 30000 个数的真因子个数,包括每个数本身}
```

```
procedure readn;
begin
  assign(f,inputfile);
  reset(F);
  readln(f,n,m);
  close(F);
end;

function max(x,y:longint):longint;{返回 x,y 中大的值}
begin
  if x>y then max:=x else max:=y;
end;

procedure output;
begin
  assign(F,outputfile);
  rewrite(F);
  writeln(f,mb);
  close(F);
end;

begin
  readn;
  mb:=0; fn:=0;
  s:=n-1;
  while s<m do begin
    t:=s+30000;
    if t>m then t:=m;
    fillchar(a,sizeof(a),0);
    a[1]:=1;{为方便统计，设 1 包含两个因子（均为 1）}
    for j:=1 to trunc(sqrt(t)) do  {枚举不同因子}
      for i:=max(j+1, (s+1) div j) to t div j do inc(a[i*j-s],2);
        {需要统计的各因子的倍数}
    for j:=trunc(sqrt(s+1)) to trunc(sqrt(t)) do inc(a[j*j-s]);
    {统计完全平方数}
    for i:=1 to t-s do{将本区间的统计结果与已知最优值比较、取优}
      if a[i]>fn then begin
        fn:=a[i];
```

```
        mb:=i+s;
      end;
    s:=t;
  end;
  output;
end.
```

3 动态规划的方法只作为理论讨论，不具可行性。

```
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q-,R+,S+,T-,V+,X+}
{$M 16384,0,655360}
program chain;
const maxn=75;
      inputfile='input.txt';
      outputfile='output.txt';
type
  word=string[maxn];{定义单词类型}
var
  f:text;
  wd,prev,ans_wd:word;{wd——当前单词；prev——前驱单词；ans_wd——最长词链}
  now,max,i,l:integer;
 {now——包括当前单词的最长词链；l——前驱单词的长度；max——最长词链的长度}

procedure output(st:word);
{将 st 串按小写字母输出}
var i:integer;
begin
  for i:=1 to length(st) do
    if st[i]=upcase(st[i]) then write(F,chr(ord(st[i])+32))
      else write(f,st[i]);
  writeln(F);
end;

begin
  assign(f,inputfile);reset(F);
  prev:='';{前驱单词清空}
  l:=0;{前驱单词长度置零}
  repeat
    readln(F,wd);{读入一个单词}
    if wd='.' then break;{文件结束标志}
```

```
    now:=0;{前缀词链个数清零}
    for i:=1 to length(wd) do
      if i<=l then begin
        if prev[i]=upcase(wd[i]){含有一个前缀单词}
          then begin inc(now);
                     wd[i]:=upcase(wd[i]);{用大写字母标记前缀的位置}
               end else if prev[i]<>wd[i] then break;
              {与前驱单词出现分歧，跳出循环}
      end else break;{大于前驱单词的长度，跳出循环}
    wd[length(wd)]:=upcase(wd[length(wd)]);{标记末字母，表示为本身的前缀}
    l:=length(wd);{递推给下一个单词}
    prev:=wd;
    if now>max then begin{保留最长的词链}
      max:=now;
      ans_wd:=wd;
    end;
  until true=false;
  close(F);
  assign(F,outputfile);rewrite(f);
  {输出结果}
  for i:=1 to length(ans_wd) do
    if ans_wd[i]=upcase(ans_wd[i]) then output(copy(ans_wd,1,i));
  close(F);
end.
```

4 问题四 IOI'99《机场跑道》，程序 land_1.pas 中实际上也用到了剪枝条件，由于效果不很明显，因此没有着重讨论。land_3.pas 仅仅是在 land_2.pas 的基础上增加了两个剪枝条件，其余完全相同，因此这里仅给出 land_3.pas。

```
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q-,R-,S+,T-,V+,X+}
{$M 2048,0,655360}
program land_1;
const maxn=700;
      last=100; {一次性读入的区域列数（东西方向）}
      inputfile='land.inp';
      outputfile='land.out';
type
  arr=array[0..maxn] of integer;
  crr=array[0..10] of longint;
  maptype=array[1..maxn] of ^arr;
```

```
var
  f:text;
  u,v,c,i,j:integer;
  xmin,ymin,xmax,ymax:integer; {机场所在区域的信息}
  map:maptype;{地图}
  area,maxarea:longint; { (area---当前的最大面积; maxarea---最大面积的上限;) }
  Buf:array[1..8191] of Char;  { 8K buffer; 缓冲区 }
  len:array[1..maxn] of crr;
  wide:crr;

procedure get(left,right:integer); {读入地图的 left 列至 right 列}
var i,j,k,no:integer;
begin
  reset(F);
  for i:=v downto 1 do begin
    readln(F);
    for j:=1 to left-1 do read(f,k);
    for no:=left to right do
      if no<=u then read(f,map[no]^[i]) else break;
  end;
end;

procedure done(stl,edl:integer);
{求解以 stl 为西面界限且其东面界限不超过 edl 中的最大区域}
var i,j,k,k0:integer;
    min,max:integer;
    maxw:longint;
begin
  if longint(edl-stl+1)*longint(v)<=area then exit;
  fillchar(len,sizeof(len),0);
  for i:=1 to v do
    for j:=0 to c do begin{求各行中包括首位置的高度在范围[min,max]内的区域}
      k:=stl;
      min:=map[stl]^[i]-j;max:=min+c;
      repeat
        inc(k);
      until (k>edl) or (map[k]^[i]>max) or (map[k]^[i]<min);
      len[i,j]:=k-stl;
```

```
    end;
  maxw:=v;
  for i:=1 to v do begin
    dec(maxw); {当前宽度}
    for k:=0 to c do begin
    {求包括 i 行首位置在内的最低高度(i 行首位置高度减 k)一定的区域的最大面积}
      wide[k]:=len[i,k]; j:=i;{北界限}
      repeat
        if wide[k]*maxw<=area then break;
        if wide[k]*(longint(j)-longint(i)+1)>area then begin
          area:=wide[k]*(longint(j)-longint(i)+1);
          xmin:=st1;xmax:=xmin+wide[k]-1;
          ymin:=i;ymax:=j;
        end;
        inc(j); {北界限累加}
        if j>v then break;
        k0:=map[st1]^[j]-map[st1]^[i]+k;
        if (k0<0) or (k0>c) then break;
        if len[j,k0]<wide[k] then wide[k]:=len[j,k0];
      until true=false;
    end;
  end;
end;

begin
  area:=1;
  xmin:=1;xmax:=1; ymin:=1;ymax:=1;
  {区域信息初始化}
  assign(f,inputfile);
  SetTextBuf(F, Buf);
  reset(F);
  readln(F,u,v,c);
  if 100>u then maxarea:=longint(v)*longint(u) else maxarea:=longint(v)*100;
  for i:=1 to 100 do new(map[i]);
  get(1,100);
  for i:=1 to u-99 do begin
    done(i,i+99);
    dispose(map[i]);{释放空间}
```

```
    if area=maxarea then break;{已经达到了最大面积的上限，跳出循环}
    if (i mod last=1) and (i+last+99<=u) then begin{读入下一轮数据}
      for j:=i+100 to i+last+99 do new(map[j]);
      get(i+100,i+last+99);
    end;
  end;
  for i:=(u div last*last)+1 to u do new(map[i]);
  get((u div last*last)+1,u); {读入剩余数据}
  for i:=u-98 to u do if i>0 then done(i,u);
  close(F);
  assign(f,outputfile);rewrite(F);
  writeln(f,area);
  writeln(f,xmin,' ',ymin,' ',xmax,' ',ymax);
  close(F);
end.

{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q-,R-,S+,T-,V+,X+}
{$M 2048,0,655360}
program land_3;
const maxn=700;
      last=100;{一次性读入的区域列数（东西方向）}
      inputfile='land.inp';
      outputfile='land.out';
type
  arr=array[0..maxn] of integer;
  maptype=array[1..maxn] of ^arr;
var
  f:text;
  u,v,c,i,j:integer;
  xmin,ymin,xmax,ymax:integer;{机场所在区域的信息}
  map:maptype;{地图}
  min,max:array[1..maxn] of integer;{分别存储一定区域内的最小高度、最大高度}
  area,maxarea:longint;{（area---当前的最大面积；maxarea---最大面积的上限；）}
  Buf:array[1..8191] of Char;  { 8K buffer；缓冲区 }

procedure get(left,right:integer);{读入地图的 left 列至 right 列}
var i,j,k,no:integer;
begin
```

```
  reset(F);
  for i:=v downto 1 do begin
    readln(F);
    for j:=1 to left-1 do read(f,k);
    for no:=left to right do
      if no<=u then read(f,map[no]^[i]) else break;
  end;
end;

procedure done(st1,ed1:integer);
{求解以 st1 为西面界限且其东面界限不超过 ed1 中的最大区域}
var i,j,k,w,l:integer; {w---当前区域沿东西向的正方形数目}
    minall,maxall:integer;
begin
  if longint(ed1-st1+1)*longint(v)<=area then exit;
  for i:=1 to v do min[i]:=maxint;
  for i:=1 to v do max[i]:=-maxint;
  w:=0;{沿东西方向的长度}
  l:=v;{沿南北方向的最大宽度}
  for i:=st1 to ed1 do begin{区域的东面界限}
    for j:=1 to v do begin
      if map[i]^[j]<min[j] then min[j]:=map[i]^[j];
      {j 行中从 st1 列至 i 列中的最小高度}
      if map[i]^[j]>max[j] then max[j]:=map[i]^[j];
      {j 行中从 st1 列至 i 列中的最大高度}
    end;
    inc(w);
    if longint(ed1-st1+1)*longint(l)<=area then break;{剪枝条件 1}
    if longint(w)*longint(l)<=area then continue;     {剪枝条件 2}
    {若可能得到的最大面积仍不能超过当前的最大面积}
    l:=0;{重新计算 L 值}
    for j:=1 to v do begin{区域的北面界限}
      k:=j+1;{区域的南面界限}
      minall:=maxint;maxall:=-maxint;
      repeat
        dec(k);
        if k=0 then break;
        if min[k]<minall then minall:=min[k];
```

```
        if max[k]>maxall then maxall:=max[k];
        {在从 k 到 j 的区域中高度的最小、最大值分别为 minall,maxall}
      until maxall-minall>c;
      {所得的区域是南北方向上的跨度为[k+1,j]}
      if j-k>l then l:=j-k;{更新 L 值}
      if longint(w)*longint(j-k)>area then begin
        area:=longint(w)*longint(j-k);
        xmin:=stl; xmax:=i;
        ymin:=k+1; ymax:=j;
      end;
    end;
  end;
end;

begin
  area:=1;
  xmin:=1;xmax:=1; ymin:=1;ymax:=1;
  {区域信息初始化}
  assign(f,inputfile);
  SetTextBuf(F, Buf);
  reset(F);
  readln(F,u,v,c);
  if 100>u then maxarea:=longint(v)*longint(u) else maxarea:=longint(v)*100;
  for i:=1 to 100 do new(map[i]);
  get(1,100);
  for i:=1 to u-99 do begin
    done(i,i+99);
    dispose(map[i]);{释放空间}
    if area=maxarea then break;{已经达到了最大面积的上限，跳出循环}
    if (i mod last=1) and (i+last+99<=u) then begin{读入下一轮数据}
      for j:=i+100 to i+last+99 do new(map[j]);
      get(i+100,i+last+99);
    end;
  end;
  for i:=(u div last*last)+1 to u do new(map[i]);
  get((u div last*last)+1,u); {读入剩余数据}
  for i:=u-98 to u do if i>0 then done(i,u);
  close(F);
```

```
  assign(f,outputfile);rewrite(F);
  writeln(f,area);
  writeln(f,xmin,' ',ymin,' ',xmax,' ',ymax);
  close(F);
end.
```

【参考书目】

1.《实用算法分析与程序设计》 吴文虎 王建德 电子工业出版社

2.《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》 吴文虎 王建德
清华大学出版社

3.《数据结构》（第二版） 严蔚敏 吴伟民 清华大学出版社