国际信息学奥林匹克竞赛指导

实用算法的分析与程序设计

吴文虎 王建德 编著

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
URL:http://www.phei.co.cn

实用算法的分析与程序设计

吴文虎　王建德　编著

電子工業出版社
Publishing House of Electronics Industry

内容简介

本书总结了历届国际奥林匹克竞赛(IOI)的试题特点及我国参赛选手的培训经验。书中许多例题取自历届大赛的试题及中国队选手的训练题目，针对问题讲解了解题的关键思路及如何灵活运用有关的算法知识。

书　　名：实用算法的分析与程序设计
编 著 者：吴文虎　王建德
责任编辑：宋玉升
特约编辑：林波
责任校对：刘丽
排版制作：雄县电脑信息服务有限公司
印 刷 者：北京天竺颖华印刷厂
装 订 者：三河市金马印装有限公司
出版发行：电子工业出版社出版、发行　URL：http://www.phei.com.cn
　　北京市海淀区万寿路173信箱　邮编100036　发行部电话：68214070
经　　销：各地新华书店经销
开　　本：787×1092 1/16　印张：22.5　字数：500千字
版　　次：1998年1月第一版　1998年1月第一次印刷
书　　号：ISBN 7-5053-4402-1/TP·2036
定　　价：28.00元

前　言

国际信息学奥林匹克（International Olympiad in Informatics，简称IOI）是联合国教科文组织倡议和发起的，在世界范围对青少年普及计算机知识的活动，也是世界五大学科竞赛之一。

IOI的宗旨是，通过竞赛形式，对有才华的青少年起到激励作用，促其能力得以发展；让青少年彼此建立联系，推动知识与经验的交流，促进合作与了解；宣传信息学这一新兴学科，给学校这一类课程注入活力，增加动力，启发新的思路；建立起教育工作者与专家之间的国际联系，推进学术思想的交流。

中国是首届参赛的十三个国家之一。从1989年到1996年，IOI已举办了8届，参赛国家和地区近60个。中国每年派出代表队，共有31人次参赛，并全部获奖，累计金牌17块、银牌6块、铜牌8块。总分成绩多次名列世界之冠。信息学竞赛是智力与用计算机解题能力的较量。中国学生在历届竞赛中显露出自己的才华，带动了一大批有志于攀登科学高峰的青少年投身于学习计算机知识、掌握计算机技能的热潮。十三年前小平同志以伟大政治家的远见卓识提出："计算机的普及要从娃娃做起"，今天重温，倍感亲切。让中国的孩子们从小就接触电脑，学用电脑，是落实"科教兴国"战略的一项重要举措。

计算机是现代科技的基础，它的出现和发展把社会生产力水平提到前所未有的高度，开创了新的信息时代。在这样一个时代，电脑与电脑文化使人类社会欣欣向荣，充满生机。人们越来越认识到，计算同语言一样，是人们每时每刻都离不开的东西。现代计算已经成为重要的科学研究方法，其重要性可与理论研究和实验研究并驾齐驱，电脑成了"人类通用的智力工具"，掌握电脑文化已经成为发挥聪明才智和创造能力的一个素质条件。在新的世纪，每个人都将经历电脑文化的洗礼，不管你愿意也罢，不愿意也罢。

参加奥赛的青少年在学用电脑方面更上了一层楼，从小培养他们的兴趣与能力，使他们在进入高等院校学习时如鱼得水，有的人取得了博士学位。

为进一步推动计算机普及工作，我们总结了指导奥赛选手学习的经验，积累了一些资料，编写了这本书，起名为《实用算法的分析与程序设计》。书中的许多例题，取自历届世界大赛的试题和中国队选手训练的题目。众所周知，计算机解题的核心是算法设计，而设计算法的关键在于思路。本书针对一些问题，既介绍基本算法，更着力分析解题的关键思路。我们不提倡死套算法，而是针对不同问题，灵活地运用有关算法知识，在这一过程中让学生去发挥自己的创造性，达到高效、简捷和最优。我们希望这本书起到抛砖引玉的作用，希望读者的做法和解法会比书中所讲的更好。如果是这样，那么我们构思这本书的初衷："用电脑帮助人脑开发"就算达到了。

中国计算机学会普及委员会主任
国际信息学奥林匹克中国队总教练
吴文虎　　1997.4.5.

目　录

第一章　基础算法

算法是一组（有限个）规则，它为某个特定问题提供了解决问题的运算序列。通俗点说，就是计算机解题的过程。在这个过程中，无论是形成解题思路还是编写程序，都是在实施某种算法。前者是推理实现的算法，后者是操作实现的算法。

计算机解题的核心是算法设计。一个算法应该具有以下五个重要特征：

(1) 有穷性　一个算法必须保证执行有限步之后结束；

(2) 确切性　算法的每一步骤必须确切定义；

(3) 输入　一个算法有0个或多个输入，以刻划运算对象的初始情况。所谓0个输入是指算法本身定出了初始条件；

(4) 输出　一个算法有一个或多个输出，以反映对输入数据加工后的结果。没有输出的算法是毫无意义的；

(5) 能行性　算法原则上能够精确地运行，而且人们用笔和纸做有限次即可完成。

下面，我们对构成算法所依据的一些基本方法（公式、方案、原则），即解题思路展开讨论，对于读者来说，为了获得一个既有效又优美的算法，必须了解一些基本的常用的算法设计思路。

§1.1　递推法

有一类试题，每相邻两项数之间的变化有一定的规律性，我们可将这种规律归纳成如下简捷的递推关系式：

$$F_n = g(F_{n-1})$$

这就在数的序列中，建立起后项和前项之间的关系。然后从初始条件（或最终结果）入手，一步步地按递推关系式递推，直至求出最终结果（或初始值）。很多程序就是按这样的方法逐步求解的。如果对一个试题，我们要是能找到后一项数与前一项数的关系并清楚其起始条件（最终结果），问题就好解决，让计算机一步步算就是了。让高速的计算机从事这种重复运算，可真正起到“物尽其用”的效果。

递推分倒推法和顺推法两种形式。一般分析思路：

if 求解初始条件 F_1

then begin {倒推}

由题意（或递推关系）确定最终结果 F_n;

求出倒推关系式 $F_{i-1}=g'(F_i)$;

i=n; {从最终结果 F_n 出发进行倒推}

while 当前结果 F_i 非初始值 F_1 do 由 $F_{i-1}=g(F_i)$ 倒推前项;

输出倒推结果 F_1 和倒推过程;

end {then}

else begin {顺推}

由题意(或递推关系)确定初始值 F1(边界条件);

求出顺推关系式 $F_i=g(F_{i-1})$;

i=1;{由边界条件 F_1 出发进行顺推}

while 当前结果 F_i 非最终结果 F_n do 由 $F_i=g(F_{i-1})$ 顺推后项;

输出顺推结果 F_n 和顺推过程;

end; {else}

一、倒推法

所谓倒推法，就是在不知初始值的情况下，经某种递推关系而获知问题的解或目标，再倒过来，推知它的初始条件。因为这类问题的运算过程是一一映射的，故可分析得其递推公式。然后再从这个解或目标出发，采用倒推手段，一步步地倒推到这个问题的初始陈述。

下面举例说明。

【例 1】贮油点

一辆重型卡车欲穿过 1000 公里的沙漠，卡车耗油为 1 升/公里，卡车总载油能力为 500 公升。显然卡车装一次油是过不了沙漠的。因此司机必须设法在沿途建立几个储油点，使卡车能顺利穿越沙漠，试问司机如何建立这些贮油点？每一贮油点应存多少汽油，才能使卡车以消耗最少汽油的代价通过沙漠？

算法分析:

编程计算及打印建立的贮油点序号，各贮油点距沙漠边沿出发的距离以及存油量。

No.	distance(k.m.)	oil(litre)
1	× ×	× ×
2	× ×	× ×
3	× ×	× ×
…	……	……

设 dis[i]－第 i 个贮油点至终点(i = 0)的距离;

oil [i]－第 i 个贮油点的存贮油量;

我们可以用倒推法来解决这个问题。从终点向始点倒推，逐一求出每个贮油点的位置及存油量。图 1.1 － 1 表示倒推时的返回点。

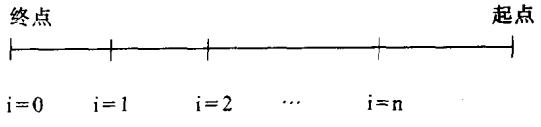

图 1.1 - 1

从贮油点 i 向贮油点 i+1 倒推的策略是，卡车在点 i 和点 i+1 间往返若干次。卡车每次返回 i+1 处时正好耗尽 500 公升汽油，而每次从 i+1 处出发时又必须装足 500 公升汽油。两点之间的距离必须满足在耗油最少的条件下使 i 点贮足 i*500 公升汽油的要求 $(0\leqslant i\leqslant n-1)$。具体地讲，第一个贮油点 i=1 应距终点 i=0 处 500km，且在该处贮藏 500

公升汽油，这样才能保证卡车能由 i＝1 处到达终点 i＝0 处，这就是说

$$dis[1]=500 \qquad oil[1]=500;$$

为了在 i=1 处贮藏 500 公升汽油，卡车至少从 i=2 处开两趟满载油的车至 i=1 处。所以 i=2 处至少贮有 2*500 公升汽油，即 oil[2]＝500*2＝1000。另外，再加上从 i=1 返回至 i=2 处的一趟空载，合计往返 3 次。三次往返路程的耗油量按最省要求只能为 500 公升，即 $d_{12}=500/3$km，

$$dis[2]=dis[1]+d_{12}=dis[1]+500/3$$

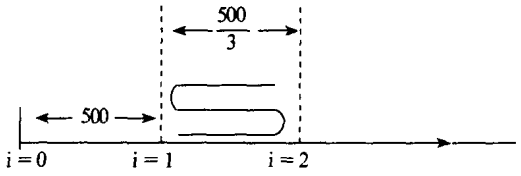

图 1.1-2

为了在 i=2 处贮存 1000 公升汽油，卡车至少从 i=3 处开三趟满载油的车至 i=2 处。所以 i=3 处至少贮有 3*500 公升汽油，即 oil[3]＝500＊3＝1500。加上 i＝2 至 i＝3 处的二趟返程空车，合计 5 次。路途耗油量亦应 500 公升，即 $d_{23}=500/5$，

$$dis[3]=dis[2]+d_{23}=dis[2]+500/5;$$

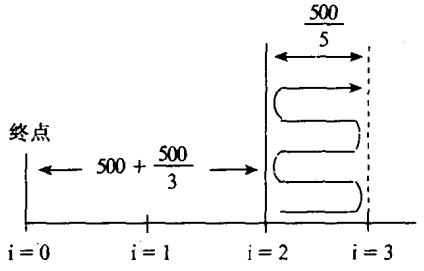

图 1.1-3

依次类推，为了在 i=k 处贮藏 k*500 公升汽油，卡车至少从 i=k＋1 处开 k 趟满载车至 i＝k 处，即 oil[k＋1]＝[k＋1]＊ 500＝oil[k]＋500，加上从 i＝k返回 i＝k＋1 的 k－1 趟返程空间，合计 2k － 1 次。这 2k － 1 次总耗油量按最省要求为 500 公升，即 $d_{k,k+1}=500/(2k-1)$，

$$dis[k+1]=dis[k]+d_{k,k+1}$$
$$=dis[k]+500/(2k-1);$$

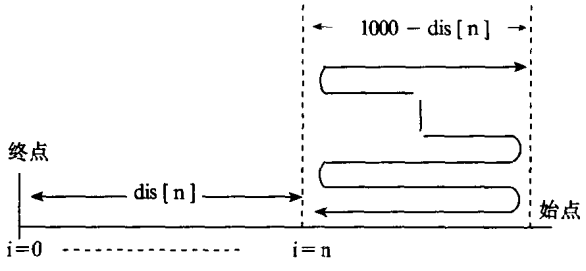

图 1.1-4

最后，i=n 至始点的距离为 1000－dis[n]，oil[n]＝500*n。为了在 i=n 处取得 n*500 公升汽油，卡车至少从始点开 n＋1 次满载车至 i=n，加上从 i=n 返回始点的 n 趟返程空车，合计 2n＋1 次，2n＋1 趟的总耗油量应正好为 (1000－dis[n]) * (2n＋1)，即始点藏油为 oil[n]＋(1000－dis[n]) * (2n＋1)。

下面为程序题解：

```
program oil_lib;
var
  k: integer; {贮油点位置序号}
  d,          {累计终点至当前贮油点的距离}
  d1: real;   {i=n 至始点的距离}
  oil, dis: array [1.. 10]  of real;
  i: integer; {辅助变量}
begin
  writeln ('NO.',' distance (k. m)': 30,' oil(1.)': 80);
  k: = 1;
  d:=500;   {从 i=1 处开始向始点倒推}
  dis[1]:=500;
  oil[1]:=500;
  repeat
    k: =k+1;
    d: =d+500/(2*k-1);
    dis[k]: = d;
    oil[k]: = oil[k-1] + 500;
    until d>=1000;
  dis[k]:=1000; {置始点至终点的距离值}
  d1:=1000-dis[k-1]; {求 i=n 处至始点的距离}
  oil[k]: =d1*(2*k+1) + oil[k-1]; {求始点藏油量}
  for i:=0 to k do   {由始点开始，逐一打印始点至当前贮油点的距离和藏油量}
     writeln (i, 1000-dis[k-i]: 30, oil[k-i]: 80);
end. {main}
```

二、顺推法

倒推法的逆过程是顺推法，即由边界条件出发，通过递推关系式推出后项值，再由后项值按递推关系式推出再后项值……，依次递推，直至从问题初始陈述向前推进到这个问题的解为止。

下面举例说明。

【例 2】实数数列

一个实数数列共有 N 项，已知

$$a_i=(a_{i-1}-a_{i+1})/2+d,\ (1<i<N)\ (N<60)$$

键盘输入 N，d，a_1，a_n，m，输出 a_m。

输入数据均不需判错。

算法分析:

该题的数学味颇浓。解题前需耐下心来，对公式

$$A_i=(A_{i-1}-A_{i+1})/2+d \qquad (1<i<N)\quad (n<60)$$

作一番推敲，探讨其数值变换规律。不然的话，会无从入手。

令 $X=A_2$ $S_2[i]=(P_i, Q_i, R_i)$ 表示 $A_i=P_iX+Q_iD+R_iA_1$;

我们可以根据

$$A_i=A_{i-2}-2A_{i-1}+2D$$
$$=P_iX+Q_iD+R_iA_1$$

推出公式

$$P_iX+Q_iD+R_iA_1=(P_{i-2}-2P_{i-1})X+(Q_{i-2}-2Q_{i-1}+2)D+(R_{i-2}-2R_{i-1})A_1$$

比较等号两端 X、D 和 A_1 的系数项，可得

$$P_i=P_{i-2}-2P_{i-1}$$
$$Q_i=Q_{i-2}-2Q_{i-1}+2$$
$$R_i=R_{i-2}-2R_{i-1}$$

加上两个边界条件

$P_1=0$ $Q_1=0$ $R_1=1$ $(A_1=A_1)$

$P_2=1$ $Q_2=0$ $R_2=0$ $(A_2=A_2)$

根据 P_i、Q_i、R_i 的递推式，可以计算出

$S_2[1]=(0, 0, 1)$;

$S_2[2]=(1, 0, 0)$;

$S_2[3]=(-2, 2, 1)$;

$S_2[4]=(5, -2, -2)$;

$S_2[5]=(-12, 8, 5)$;

…… …… …… …… …… …… …… …… …… ……

$S_2[i]=(P_i, Q_i, R_i)$;

…… …… …… …… …… …… …… …… …… ……

$S_2[N]=(P_N, Q_N, R_N)$;

有了上述基础，A_M 便不难求得。有两种方法:

1. 由于 A_N、A_1 和 P_N、Q_N、R_N 已知，因此可以先根据公式

$$A_2=A_N-Q_ND-R_NA_1/P_N$$

求出 A_2。然后将 A_2 代入公式

$$A_3=A_1-2A_2+2D$$

求出 A_3。然后将 A_3 代入公式

$$A_4=A_2-2A_3+2D$$

求出 A_4。然后将 A_4 代入公式

…… …… …… …… …… …… …… …… …… …… …… …… …… …… …… …… ……

求出 A_{i-1}。然后将 A_{i-1} 代入公式

$A_i = A_{i-2} - 2A_{i-1} + 2D$

求出 A_i。依次类推，直至递推至 A_M 为止。

上述算法的缺陷是，由于 A_2 是两数相除的结果，而除数 P_N 递增，因此精度误差在所难免，以后的递推过程又不断地将误差扩大，以至当 M 超过 40 时，求出的 A_M 明显偏离正确值。显然，这种算法虽简单但不可靠。

2．我们令 $A_2=A_2$，$A_3=X$，由 $S_3[i]=(P_i, Q_i, R_i)$ 表示 $A_i=P_iX+Q_iD+R_iA_2$（$i \geqslant 2$）可计算得出

$S_3[2]=(0, 0, 1)=S_2[1]$;

$S_3[3]=(1, 0, 0)=S_2[2]$;

$S_3[4]=(-2, 2, 1)=S_2[3]$;

$S_3[5]=(5, -2, -2)=S_2[4]$;

…… …… …… …… …… …… …… …… …… …… …… …… …… …… …… …… ……

$S_3[i]=(\cdots\cdots\cdots)=S_2[i-1]$;

…… …… …… …… …… …… …… …… …… …… …… …… …… …… …… …… ……

$S_3[N]=(\cdots\cdots\cdots)=S_2[N-1]$;

再令 $A_3=A_3$，$A_4=X$，由 $S_4[i]=(P_i, Q_i, R_i)$ 表示 $A_i=P_iX+Q_iD+R_iA_3$（$i \geqslant 3$）可计算得出

$S_4[3]=(0, 0, 1)=S_3[2]=S_2[1]$;

$S_4[4]=(1, 0, 0)=S_3[3]=S_2[2]$;

$S_4[5]=(-2, 2, 1)=S_3[4]=S_2[3]$;

…… …… …… …… …… …… …… …… …… …… …… …… …… …… …… …… ……

$S_4[i]=(\cdots\cdots\cdots)=S_3[i-1]=S_2[i-2]$;

…… …… …… …… …… …… …… …… …… …… …… …… …… …… …… …… ……

$S_4[N]=(\cdots\cdots\cdots)=S_3[N-1]=S_2[N-2]$;

依次类推，我们可以发现一个有趣的式子：

$A_N=P_{N-i+2}*A_i+Q_{N-i+2}*D+R_{N-i+2}*A_{i-1}$，即

$$A_i=(A_N-Q_{N-i+2}*D-R_{N-i+2}*A_{i-1})/P_{N-i+2}$$

我们从已知量 A_1 和 A_N 出发，依据上述公式顺序递推 A_2、A_3、…、A_M。由于 P_{N-i+2} 递减，因此最后得出的 A_M 要比第一种算法趋于精确。

下面给出程序题解：

```
program ND1P4;
const
  maxn     =60;
var
```

```
  n, m, i   : integer;
  d         : real;
  list      : array [1..maxn] of real;          {list[i]——对应 a_i}
  s         : array [1..maxn, 1..3] of real;    {S[i, 1]——对应 P_i;
                                                 S[i, 2]——对应 Q_i;
                                                 S[i, 3]——对应 R_i}

procedure init;
  begin
    write('n m d =');
    readln (n, m, d);              {输入项数、输出项序号和常数}
    write('a1 a', n ,' =');
    readln (list[1], list[n]);     {输入 a_1 和 a_n}
  end; {init}

procedure solve;
  begin
    s[1, 1] :=0; s[1, 2] :=0; s[1, 3] :=1;
  {求递推边界 (P_1, Q_1, R_1)和(P_2, Q_2, R_2)}
    s[2, 1] :=1; s[2, 2] :=0; s[2, 3] :=0;
    {根据公式 P_i ← P_i-2 -2*P_i-1}
    {Q_i ← Q_i-2 -2*Q_i-1}
    {R_i ← R_i-2 -2*R_i-1}
    {递推 (P_3, Q_3, R_3)…… P_n, Q_n, R_n)}
    for i:=3 to n do
      begin
        s[i, 1] :=s[i-2, 1]-2*s[i-1, 1];
        s[i, 2] :=s[i-2, 2]-2*s[i-1, 2]+2;
        s[i, 3] :=s[i-2, 3]-2*s[i-1, 3];
        end; {for}
  end; {solve}

procedure main;
  begin
    solve;                  {求 (P_1, Q_1, R_1)..(P_n, Q_n, R_n)}
    {根据公式 A_i=(A_n-Q_n-i+2*d-R_n-i+2*A_i-1)/P_n-i+2}
    {递推 A_2..A_m}
    for i:= 2 to m do
```

```
    list[i] :=(list[n]-s[n-i+2, 2]*d -s[n-i+2, 3] * list[i-1])/s[n-i+2, 1];
  writeln('a', m, '  =', list[m]: 20: 10);    {输出 A m}
  end; {main}

begin
  init;    {输入数据}
  main;    {递推和输出 A m}
  readln;
end. {main}
```

§1.2 贪心法

和§1.1 节中所讲的顺推法相仿，贪心法也是从问题的某一个初始解出发，向给定的目标递推。但不同的是，推进的每一步不是依据某一固定的递推式，而是做一个当时看似最佳的贪心选择，不断地将问题实例归纳为更小的相似的子问题，并期望通过所做的局部最优选择产生出一个全局最优解。

问题是，这种局部贪心的选择是否可以得出全局最优解呢？在某些情况下是可以的。例如：

【例 1】删数问题

键盘输入一个高精度的正整数 N，去掉其中任意 S 个数字后剩下的数字按原左右次序组成一个新的正整数。编程对给定的 N 和 S，寻找一种方案使得剩下的数字组成的新数最小。

输出应包括所去掉的数字的位置和组成的新的正整数。（N 不超过 240 位）

输入数据均不需判错。

算法分析：

首先我们必须注意，试题中正整数 N 的有效位数为 240 位，而计算机中整数有效位（包括符号位）充其量也不过 11 位，无论如何也达不到试题的数位要求。因此，必须采用可含 256 个字符的字串来替代整数。

以字串形式输入 N，使用尽可能逼近目标的贪心法来逐一删去其中 S 个数符，每一步总是选择一个使剩下的数最小的数符删去。之所以作出这样贪心的选择，是因为删 S 个数符的全局最优解，包含了删一个数符的子问题的最优解。

为了保证删 1 个数符后的数最小，我们按高位→低位的方向搜索递减区间。若不存在递减区间，则删尾数符；否则删递减区间的首字符，这样形成了一个新数串。然后回到串首，重复上述规则，删下一数符…… 依此类推，直至删除 S 个数符为止。

例如：N='178543'，S=4，删数过程如下：

```
          ↓
N=' 1 7 8 5 4 3'
    ↓
  ' 1 7  5 4 3'
      ↓
  ' 1 5 4 3'
      ↓
  ' 1 4 3'
  ' 1 3'
```

显然，按上述规则删去S个数符后，剩余N−S个数符组成的新数必定最小。

下面是程序题解：

```
PROGRAM Find_Min_Integer;
  USES
    Crt;
  VAR
    N                    : string; {数串}
    S, i                 : integer; {S——被删的数字个数}
  BEGIN
    clrscr;
    write('N='); readln(N); {输入数串}
    write('S='); readln(S); {输入应删的数字个数}
    while S>0 do begin {逐一删去S个数符}
      i:=1;
      while (i<length(N)) and (N[i]<=N[i+1]) do inc(i); {搜索递减区间}
      delete(N, i, 1); {删去该区间的首数符}
      dec(S);
    end; {while}
    while (length(N)>1) and (N[1]='0') do delete(N, 1, 1);
     {删去串头的无用零}
    writeln(N); {输出剩下的数码}
  END. {main}
```

读者在看了上述题解后，自然会提出这样一个问题：对一个最优化问题，我们怎样才能知道它是否适用于贪心算法求解？没有一个通用的方法。但适用于贪心策略求解的大多数问题都有两个特点：

1. 贪心选择性质——可通过做局部最优（贪心）选择来达到全局最优解

贪心策略通常是自顶向下做的。第一步为一个贪心选择，将原问题变成一个相似的、但规模更小的问题，而后的每一步都是当前看似最佳的选择。这种选择可能依赖于已作出的所有选择，但不依赖有待于做的选择或子问题的解。从求解的全过程来看，每

一次贪心选择都将当前问题归纳为更小的相似子问题，而每一个选择都仅做一次，无重复回溯过程，因此贪心法有较高的时间效率。

例如【例 1】中，设正整数 N 有 p 位。第一步选择一个使剩下的 p－1 位数最小的数符删去，把问题归纳为在 p－1 位正整数中去掉 S－1 个数字后的新数最小的子问题；第二步再选择一个使得剩下的新数最小的数符删去，又把问题归纳为在 p－2 位正整数中去掉 S－2 个数字后的新数最小的子问题…… 每次选择中，已删去的数不允许再删，而当前选择又不受将来删的数的影响。依照上述方法不断将问题归纳为更小的互为独立的子问题，直至删除 s 个数符为止。显然删数问题符合贪心选择性质。

2. 最优子结构——问题的最优解包含了子问题的最优解

例如【例 1】中，N＝'178543'，S＝4。第一步的贪心选择为删'8'，即第一个子问题的最优解为'8'；第二步的贪心选择为删'7'，即第二个子问题（第一个子问题的子问题）的最优解为'7'；继后的第三个，第四个子问题的最优解分别为'5' 和'4'。很显然，问题的最优解包含了四个子问题的最优解，因此删数问题具有最优子结构性质。

但问题是，对所有具备最优子结构的问题是否都可采用贪心策略呢？不一定。下面，我们来考察一个经典优化问题的两种变形。

【例 2】背包问题

有一个贼在偷窃一家商店时发现有 N 件物品：第 i 件物品值 V_i 元，重 W_i 磅，（$1 \leqslant i \leqslant n$），此处 V_i 和 W_i 都是整数。他希望带走的东西越值钱越好，但他的背包中最多只能装下 W 磅的东西（W 为整数）。有两种偷窃方式：

1) 0-1 背包问题

如果每件物品或被带走或被留下，小偷应该带走哪几件东西？

2) 部分背包问题

如果允许小偷可带走某个物品的一部分，小偷应该带走哪几件东西，每件东西的重量是多少？

算法分析：

两种背包问题都具有最优化结构性质：

对于 0－1 背包问题，考虑重量至多为 W 磅的最值钱的一包东西。如果我们从中去掉物品 J，余下的必须是窃贼从其余的 n－1 件物品中可带走的重量至多为 $W-W_J$ 的最值钱的一件东西；如果我们再从中去掉物品 i，余下的也必须是窃贼从其余 n－2 件可带走的、重量至多为 $W-W_j-W_i$ 的最值钱的一件东西…… ，依次类推。

对于部分背包问题，如果我们考虑从最优货物中去掉某物品 J 的重量 W_P（$W_J \geqslant W_P$），则余下的物品必须是窃贼可以从 N－1 件原有物品和物品 J 的 W_J-W_P 中可带走的、重量至多为 $W-W_P$ 的最值钱的一件东西 …… ，依次类推。

我们可采用贪心策略来解决部分背包问题：

先对每件物品计算其每镑价值 V_i/W_i，然后按每磅价值单调递减的顺序对所有物品排序。例如，总共有三件物品和一个背包。