# 目　录

## 第一部分　常用算法

第一章　基础题……1
第二章　枚举法 ……15
第三章　不同进制数的转换及应用 ……21
第四章　高精度计算 ……27
第五章　数据排序 ……32
第六章　排列和组合 ……44
第七章　递推算法 ……49
第八章　递归算法 ……57
第九章　回溯算法 ……63
第十章　贪心算法 ……74
第十一章　分治算法策略 ……81
第十二章　深度优先搜索算法 ……89
第十三章　广度优先搜索算法 ……101
第十四章　动态规划……112
　　第一节　动态规划的基本模型……112
　　第二节　动态规划与递推……116
　　第三节　历届 NOIP 动态规划试题 ……128
　　第四节　背包问题……147
　　第五节　动态规划应用举例……158
第十五章　递推关系在竞赛中的应用……180

## 第二部分　数据结构

第一章　线性表 ……189
第二章　指针与链表 ……191
第三章　栈 ……203
第四章　队列 ……213
第五章　树 ……224
第六章　图 ……252

# 第一部分　常用算法

在计算机程序设计中讨论算法的目的则是将其作为编写程序的依据，它是软件设计的基础。算法的好坏，将影响着软件的质量，因此研究算法对提高软件的质量起着很重要的作用。本章将就程序设计中常见的典型算法做一些介绍。

## 第一章　基础题

奥林匹克信息学竞赛十分强调基础知识。每年的分区联赛（NOIP）都含一些直接考核选手是否会编程的基础题，而每年的全国赛（NOI）、组队赛（CTSC）或国际赛（IOI）对灵活应用基础知识的要求也愈来愈高。因此，在平日训练中既不要因为畏难而放弃大题难题，更不要因为轻视而不屑做基础题。自信心和自知之明、夯实基础和破解难题是对立的统一。我们在做大题难题的时候，为什么效果经常不尽如人意，除了采用算法错误的原因外，更多的时候是因为细节上的一些瑕疵而导致算法的关键地方时效低下，甚至导致整个算法的时间复杂度远远高于正常情况，最为严重的后果是导致正确的算法无法实现。出现“因小失大，功亏一篑”的主观原因是好高骛远、轻视基础。因此我们应该注意从小题、中题练起，积累经验，强化内功，夯实基础。

### 【例 1】计算多项式

设多项式 exp(x)=1+x+$\frac{x^2}{2!}+\frac{x^3}{3!}+\cdots\cdots+\frac{x^i}{i!}$ ($\left|\frac{x^i}{i!}\right|\leqslant 10^{-10}$)

输入 x

输出 exp(x)的值（保留小数点后四位）

**题解**

设 s—和；t—当前项；i—x 的幂次。即 $s=t_0+t_1+t_2+\cdots\cdots+t_i$

1. 确定重复条件

   abs（t）>1e-10；

2. 确定重复体

   由 $t_i=\frac{x^{i-1}}{(i-1)!}*\frac{x}{i}=t_{i-1}*\frac{x}{i}$ 得出重复体

   i←i+1；

   t←t*$\frac{x}{i}$；

   s←s+t；

3. 设定初值

i←0; t←1; s←1;

由此得出程序：

```
var
 i，X：integer;                                          {x 的幂次和自变量}
 t，s：real;                                             {当前项和多项式的值}
 begin
  readln(x);                                             {读自变量的值}
  i：=0;  t：=1;  s：=1;                                  {赋初值}
  while abs(t)>1e-10 do                                  {若当前项未达到精度要求,则新增一项}
   begin
    i：=i+1;
    t：=t*x/i;
    s：=s+t;
   end; {while}
  writeln(' EXP(', X,' )= ', S：0：4);                    {输出多项式的值}
  readln;
 END.{main}
```

## 【例 2】计算灯的开关状态

有 N 个灯放在一排，从 1 到 N 依次顺序编号。有 N 个人也从 1 到 N 依次编号。1 号将灯全部关闭，2 将凡是 2 的倍数的灯打开；3 号将凡是 3 的倍数的灯作相反处理(该灯如为打开的， 则将它关闭；如关闭的，则将它打开)。以后的人都和 3 号一样，将凡是自己编号倍数的灯作相反处理。试计算第 N 个操作后，哪几盏灯是点亮的。(0-表示灯打开 1-表示灯关闭)

**题解**

设布尔数组 a 为 n 盏灯的状态。初始时，所有的灯打开，即 a 数组的每一个元素设为 false。然后依次将每盏灯序号的倍数作取反处理，由此得出的 a 数组的序数值即为最后的灯状态。

```
var  k，n，i，j：integer;
    a：array[1..100]of boolean;                          {N 盏灯的状态}
begin
  readln (n);                                            {读入灯的数目}
  for i：=1 to n do a[i]：=false;                         {初始时所有灯打开}
  for i：=1 to n do                                      {依次进行 n 次操作}
   begin
    j：=i;                                               {从第 i 号队员出发进行第 i 次操作}
    while j<=n do
      begin
       a[j]：=not(a[j])；j：=j+i;                        {将凡是 i 的倍数的灯作取反处理}
      end; {while}
    end; {for}
```

```
  for i：=1 to n do write(ord(a[i]));                    {输出 n 次操作后的结果}
  writeln;
end.{main}
```

**【例 3】计算今天星期几**

按照年 月 日的格式输入今天的日期。计算和输出今天是星期几的信息。

**题解**

设年、月、日为 y、m、d。按照余数公式

$(a_1+a_2+\cdots a_n)\bmod k=(a_1 \bmod k+a_2 \bmod k+\cdots a_n \bmod k)\bmod k$

我们首先计算公元 0000 至去年（y-1）间每年天数对 7 的余数，累计它们的和 y’；然后计算今年 1 月至上月（m-1）的天数对 7 的余数 m’；最后得出(y’+m’+d)mod 7 为今天的星期信息。

1.计算公元 0000 至 y-1 年对 7 的余数和 y’

闰年的天数为 366，对 7 的余数为 2；平年的天数为 365，对 7 的余数为 1。公元 0000 年至去年（y-1）含$\left\lfloor\frac{y-1}{4}\right\rfloor-\left\lfloor\frac{y-1}{100}\right\rfloor+\left\lfloor\frac{y-1}{400}\right\rfloor$个闰年。由此得出公元 0000 至 y-1 年对 7 的余数和为 y’=y-1+$\left\lfloor\frac{y-1}{4}\right\rfloor-\left\lfloor\frac{y-1}{100}\right\rfloor+\left\lfloor\frac{y-1}{400}\right\rfloor$。

2. 计算今年 1 月至上月(m-1)的总天数对 7 的余数 m’

$$m'=\begin{cases}0 & \text{上月为1,10月}\\ 1 & \text{上月为5月}\\ 2 & \text{上月为8月}\\ 3 & \text{上月为2,3,11月}\\ 4 & \text{上月为6月}\\ 5 & \text{上月为9,12月}\\ 6 & \text{上月为4,7月}\end{cases}$$

若今年为闰年(((y mod 4=0) and(y mod 100<>0))or(y mod 400=0))且月份大于 2(m>2)，则 m’=m’+1；

3.根据表达式(y’+m’+d)mod 7 的值，分情形输出今天是星期几的信息

```
var
 y，y0，m，d，m0，s，week：integer;
begin
 readln(y，m，d);                                          {读今天的日期}
 y0：=y-1;                                                 {计算去年的年号}
 y0：=y0+y0 div 4-y0 div 100+y0 div 400;        {计算公元 0000 至去年对 7 的余数和}
 case m of                                   {计算 1 月至 m-1 月的总天数对 7 的余数 m0}
```

```
   1，10：  m0：=0;
      5：  m0：=1;
      8：  m0：=2;
 2，3，11：  m0：=3;
      6：  m0：=4;
   9，12：  m0：=5;
    4，7：  m0：=6;
 end; {case}
if(((y mod 4=0)and(y mod 100<>0))or(y mod 400=0))and(m>2) then m0：=m0+1;
case（y0+m0+d）mod 7 of  {根据(y’+m’+d)mod 7 的值，分情形输出今天是星期几的信息}
   0：writeln(’Sunday’);
   1：  writeln(’Monday’);
   2：  writeln(’Tuesday’);
   3：  writeln(’Wednesday’);
   4：  writeln(’Thursday’);
   5：  writeln(’Friday);
   6：  writeln(’Saturday’);
 end; {case}
readln;
end.{main}
```

**【例 4】放球**

把 m 个球放入编号为 0，1，2，…，k-1 的 k 个盒中($m<2^k$)要求第 i 个盒内必须放 $2^i$ 只球。如果无法满足这一条件，就一个不放，求出放球的具体方案。

**题解：**

将十进制数 M 化为对应的二进制数。将第 i 位的数码乘以 $2^i$ 即为第 i($0\leqslant i\leqslant \log_2 m$)个盒子应放的球数。例如 M=29，化为二进制数是 11101，$29=2^4 4+2^3+2^2+1$，即第 0 号盒放 1 只球，1 号盒不放，2 号盒放 4 只球，3 号盒放 8 只球，4 号盒放 16 只球。设

$(A)_2=a_k a_{k-1}\cdots a_1 a_0$　$(B)_2=b_k b_{k-1}\cdots b_1 b_0$　$(C)_2=c_k c_{k-1}\cdots c_1 c_0$（$0\leqslant a_i$，$b_i$，$c_i\leqslant 1$，$0\leqslant i\leqslant k$）

turbo.Pascal 提供了五种位运算：

| | |
|---|---|
| C=not A | 按位取反（一元运算），即 $c_i=\overline{a_i}$； |
| C=A and B | 按位与，即 $a_i=b_i=1$ 时，$c_i=1$；否则 $c_i=0$； |
| C=A or B | 按位或，即 $a_i$ 和 $b_i$ 中至少有一个为 1 时，$c_i=1$；否则 $c_i=0$； |
| C=A xor B | 按位异或，即 $a_i$ 和 $b_i$ 相反时，$c_i=1$；否则 $c_i=0$； |
| C=A shl 1 | $(A)_2$ 左移 1 位，相当于乘 $2^1$； |
| C=A shr 1 | $(A)_2$ 右移 1 位，相当于整除 $2^1$； |

注意，为了保证位运算的正确性，参与位运算的变量一般为无符号整数（byte 或 word）。如果使用有符号整数（longint 或 integer），则最高位的符号位不能参与运算。例如，A=10，B=7，L=2。

C=not A=$(\overline{1010})_2=(0101)_2=5$， $C = AandB = and\begin{array}{r}1010\\0111\\0010\end{array}=2$， $C = AorB = or\begin{array}{r}1010\\0111\\1111\end{array}=15$，

$C = AxorB = xor\begin{array}{r}1010\\0111\\1101\end{array}=13$。

C=A shl l=40

C=A shr l=2

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | ← |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | →1 0 |

表 3.3.3

显然，我们可以通过(m shr 1)and 1 取出 m 的下一位二进制数码。但问题是 Pascal 不能直接计算 $2^i$， 我们通过两个函数和相应的换底公式

EXP（x）—计算 e 的 x 次幂 $e^x$；

Ln（x）—计算以 e 为底的对数；

$2^i=e^{i*\ln(2)}$ =EXP(i*ln(2))；

计算出 $2^i$。按照由底位至高位的顺序逐位取出$(m)_2$的二进制数码，乘上 $2^i$（i 为位序号），便可以得出每一个盒子放的球数。计算过程如图 3.3.2 所示：

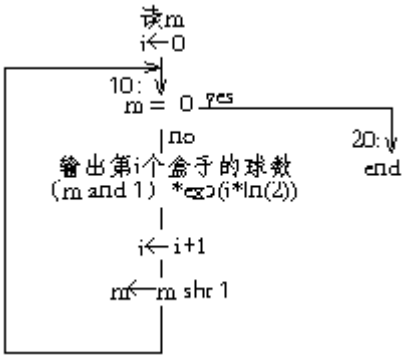



图 3.3.2

```
var
 m，i：longint;                                          {小球数和位序号}
begin
  readln(m);                                             {读小球数}
  i:=0;                                                  {从第 0 个盒子开始计算}
  while m<>0 do                                          {若目前 i 个盒子未放入所有小球，则循环}
   begin
    writeln(i,':', (m and 1)*exp(i*ln(2))：0：0);        {输出第 i 个盒子的球数}
    i:=i+1;                                              {增加一个盒子}
    m：=m shr 1;                                         {右移一位}
   end; {while}
  readln;
end.{main}
```

**【例 5】代码翻译**

输入一个以’@’结束的字符串，从左至右翻译。若下一个字符是数字 n(0≤n≤9)，表示后一个字符重复 n+1 次，不论后一个字符是否为数字；若下一个字符非数字，则表示自己。翻译后，以 3 个字符为一组输出，组与组之间用空格分开。例如’A2B5E34FGOZYWPQ59R@’，翻译成’ABB_BEE_EEE_E44_44F_GZY_WPQ_999_999_R@ ’。

输入

字符串（串长≤255）；

输出

翻译后的字符串；

**题解**

设 ch 为输入字符；

n 为组内计数器。按照三个译码为一组、组与组之间以空格分隔的输出要求，每输出一个译码，n←(n+1)mod 3。当 n=0 时，一组译码输出完毕，应输出一个空格；

c 为字符重复输出的次数。当 c>’0’时，说明字符 ch 作为译码还应继续输出。c←pred(c)，输出 ch；当 c=’0’时，说明译码 ch 输出完毕，重新输入一个字符 ch。若 ch 为数字符，则表示 ch 的后一个字符作为译码，该译码应重复 ord(ch)-ord(’0’)次，c←ch；若 ch 为字母，则表示 ch 作为译码应该输出一次；

我们在输入字符的同时，按照上述翻译规则进行译码：

```
var    c，ch：char;                                    {重复次数和输入字符}
       n：integer;                                     {组内计数器}
begin
 n：=0;  c：=’0’;                                      {组内计数器和重复次数初始化}
 read(ch);                                             {输入字符}
 if ch<>’@’                                            {若当前字符非结束符}
  then repeat
        if c>’0’                                       {若当前字符需要重复输出，则重复次数-1}
         then c：=pred(c)
         else if (ch>=’0’)and(ch<=’9’)
                  then begin                           {若输入的字符表示重复次数，则记下}
                         c：=ch;
                         read(ch);                     {输入被重复的字符}
                       end; {then}
        write(ch);                                     {输出译码}
```

```
        n：=(n+1)mod 3;                                    {计算组内计数器}
        if n=0 then write(' ');                  {若一组输出完毕，则以空格间隔}
        if c=' 0' then read(ch);{若当前字符完成了重复输出的次数，则读下一个字符}
       until ch=' @';                                      {直至读入结束符为止}
 writeln(' @');
end.{main}
```

**【例 6】字符加密**

加密规则是将输入的英文字母下推 K 个顺序后输出。加密工作直至输入一个非英文字母为止。例如 K=5

| 输入 | 输出 |
|---|---|
| ' a' | ' f' |
| ' b' | ' g' |
| …… | |
| ' y' | ' d' |
| ' Z' | ' E' |
| ' 0' | 结束 |

**题解**

设输入字符为 ch。首先我们通过布尔表达式（ch in[' a'..' z',' A'..' Z']）判断字符 ch 是否为英文字母。如果是，则通过布尔表达式（ch=upcase(ch)）确定该英文字母的大小写，因为下推 K 个顺序的计算只能在所属的大小写范围内进行。ch 下推 K 个顺序后的字母在 26 个英文字母中的顺序为 ord(ch)-ord(' A')+k)mod 26（或 ord(ch)-ord(' a')+k）mod 26），加上' A'（或' a'）的 ascll 码（ord(' A')或 ord(' a')）即为译码的 ascll 码，通过 chr 函数将其还原为译符。

```
var
 ch:char;                                                    {输入字符}
 k:integer;                                                  {下推的顺序数}
begin
 readln(k);                                                  {读下推的顺序数}
 readln(ch);                                                 {读第 1 个字符}
 while ch in[' a'..' z',' A'..' Z']do              {若该字符为英文字母，则循环}
  begin
   write(ch,' :');
   if ch=upcase(ch)                    {根据 ch 的大小写计算下推 K 个顺序后的译符}
    then ch:=chr((ord(ch)-ord(' A')+k)mod 26+ord(' A'))
    else ch:=chr((ord(ch)-ord(' a')+k)mod 26+ ord(' a'));
   writeln(ch);                                              {输出译符}
   readln(ch);                                               {读下一个字符}
  end; {while}
 end.{main}
```

**【例 7】计算路程**

一次军事演习，A、B 两队约好同一时间从相距 s(50≤s≤100)公里的各自驻地出发相向运动，A 队行进速度为 va 公里/小时(5≤va≤10)，B 队为 vb 公里/小时(4≤vb≤8)。一通迅员骑摩托车从 A 队的驻地也在同一时间为行进中的两队传递信息。摩托车的速度为 vm(30≤vm≤60)60 公里/小时，往返于两队之间，每遇一队立即折回驶向另一队。当两队距离小于 0.5 公里时，摩托车停下来不再传递信息。

输入

s,va,vb,vm

输出

通讯员跑了多少趟。( 从一队驶向另一队叫一趟)

**题解**

s——A.B 两队之间的距离，初值为两地的距离；

va，vb，vm——A 队，B 队和摩托车的行进速度；

j——通讯员所跑的趟数；

t—当前一趟摩托车的费时；

i——通讯员往返于两队的标志。$i=\begin{cases}1 & \text{通讯员从}A\text{队折回驶向}B\text{队}\\ -1 & \text{通讯员从}B\text{队折回驶向}A\text{队}\end{cases}$

i=1，通讯员从 A 队折回驶向 B 队相遇时，满足 s-vb*t=vm*t.即 t=$\frac{s}{vb+vm}$；

i=-1，通讯员从 B 队折回驶向 A 队相遇时，满足 s-va*t=vm*t.即 t=$\frac{s}{va+vm}$；

由上式可以看出，由于 t 为除法运算的结果，而 s 的计算过程中有 t 参与，因此 s 和 t 为实数类型。

```
var    va:5..10;                                  { A 队，B 队，摩托车的行进速度}
       vb:4..8;     vm:30..60;
       i，j：integer;                                          {往返标志和趟数}
       s，t：real;                                 {两队的距离和当前一趟摩托车的费时}
 begin
  readln(s,va,vb,vm);                {输入两队的距离和 A 队、B 队、摩托车的行进速度}
  i：=1；j：=0;                                            {往返标志和趟数初始化}
  repeat
    if i=1 then t：=s/(vb+vm)                      {根据往返标志计算当前一趟的时间}
           else t：=s/(va+vm);
    s:=s-(va+vb)*t;                                                  {计算两队距离}
    j：=j+1;                                                          {累计趟数}
    i：=i*(-1);                                                      {往返标志取反}
  until s<=0.5;                                     {直至两队距离小于 0.5 公里为止}
  writeln (’ j=’，j);                                                {输出趟数}
 end.{main}
```

**【例 8】计算数列**

求 1/1+1/2+2/3+3/5+5/8+…前 n 项（n≤50）的和（保留小数点后 2 位）。

**题解**

设 $e=t_1+t_2+\cdots t_i+\cdots t_n$，其中 $t_i=\frac{a_i}{b_i}$（1≤i≤n）。由数列的特征可以看出，$a_i=b_{i-1}$，$b_i=a_{i-1}+b_{i-1}$。由于 n≤50，因此 $a_i$，$b_i$ 和 e 的数值超过了标准的整数类型和实数类型的范围。不得不通过{$n+}启动浮点运算，并将 $a_i$，$b_i$ 和 e 的数据类型设为 extended，使得其精度保持在 19 位。

```
{$N+}                                                    {启动浮点运算}
var i，n：interger;                                      {循环变量和项数}
    a，b，c，e：extended;  {数列的和为 e；当前项的分子为 a，分母为 b；c 为辅助变量}
begin
  readln(n);                                             {读项数}
  a：=0；b：=1；e：=0;                    { 当前项的分子、分母和数列的和初始化}
  for i：=1 to n do                                      {累计每一项}
    begin
     c：=b；b：=a+b；a：=c;                              {计算第 i 项的分子和分母}
     e：=e+a/b;                                          {将第 i 项计入数列和}
    end; {for}
  writeln(e：0：2);                                      {输出数列和}
end.{main}
```

**【例 9】从 m 个不同数中取 n 个不同数的全部组合**

输入 m，n(1≤n≤m≤100)

输出所有组合

**题解**

设组合为 $c_1\cdots c_n$。要求组合中的 n 个数按照递增顺序排列。$c_n$ 最大为 m，$c_{n-1}$ 最大为 m-1，…，$c_1$ 最大为 m-n+1，即 $c_j$≤m-n+j。计算组合 c 的过程如下：

1. 初始时，$c_1=1$，$c_2=2$，…，$c_n=n$，即 $c_1\cdots c_n$ 为最小。
2. 从 $c_n$ 出发向右扫描。若 $c_{i+1}\cdots c_n$ 都达到了最大值（$c_j=m-n+j$，j=i+1…n），$c_i<m-n+i$，则 $c_1\cdots c_{i-1}$ 保持不变，$c_i=c_i+1$，$c_{i+1}=c_i+1$，$c_{i+2}=c_{i+1}+1$，…，$c_n=c_{n-1}+1$。输出组合 $c_1\cdots c_n$。例如 1…6 中取 3 个数的组合 256。由于 $c_1=2<4$，因此可按照上述方法计算和输出下一个组合 345。
3. 返回 2、计算下一个组合，直至 $c_1\cdots c_n$ 都达到了最大值为止。

例如从 1、2、3、4、5 中取 3 个不同数，按照上述算法可得出十个组合：

1 2 $\underline{3}$　1 2 $\underline{4}$　1 $\underline{2}$ 5　1 3 $\underline{4}$　1 $\underline{3}$ 5

$\underline{1}$ 4 5　2 3 $\underline{4}$　2 $\underline{3}$ 5　$\underline{2}$ 4 5　3 4 5　（$\underline{c_i}$<m-n+i）

显然当 $c_1\cdots c_n$ 都达到了最大值时，$c_1=m-n+1$。由此得出算法的终止标志 $c_1=m-n+1$。

```
Var  i,j,k:integer;
     m,n:1..1000;                                        {问题规模}
     b:array[0..100] of integer;                         {组合}
```

```
begin
 repeat
    readln(m,n);                                                    {读问题规模}
 until m>n;
 for i:=1 to n do b[i]:=i;                                          {设置和输出初始组合}
for i:=1 to n do write(b[i]:5);
 while b[1]<m-n+1 do                                  {若 c1…cn 未全部达到最大值，则循环}
  begin
   j:=n;                                          {由右而左搜索第 1 个未达到最大值的元素 j}
   while b[j]=m-n+j do j:=j-1;
   b[j]:=b[j]+1;                                                    {调整元素 j…元素 n}
   for i:=j+1 to n do b[i]:=b[i-1]+1;
   for i:=1 to n do write(b[i]:5);                                  {输出当前组合}
   writeln;
  end; {while}
end.{main}
```

**【例 10】计算成等差数列的素数**

求出 2~n（n≤50）之间长度最长、成等差数列的素数。

例如：2~50 之间的奇数

2　3　5　7　11 13　17　19　23　29　31　37　41　43　47

公差为 1：　2，　3　　长度为 2

公差为 2：　3　5　7　　长度为 3

输入　n

输出：成等差数列的素数

**题解**

我们首先通过筛选法计算素数集合。设 s 为筛；primes 为素数集合。初始时，s=[2..n]，primes=[]。选取 s 筛中的最小数，作为素数进入 primes 集合，并将该素数的倍数从 s 筛中划去；然后再选取 s 筛中最小数为素数，并将 s 筛中该素数的倍数划去，…，直至 s 筛空为止。此时 primes 集合即存储了 2..n 中的素数。例如表 3．4．1 给出了计算 2—10 间素数的过程：

| 动作 | S 集合 | Primes 集合 |
|---|---|---|
| 初始状态 | [2，3，4，5，6，7，8，9，10] | [] |
| 2 进入 Primes 集合，将 2 的倍数从 s 筛中划去 | [3，5，7，9] | [2] |
| 3 进入 Primes 集合，将 3 的倍数从 s 筛中划去 | [5，7] | [2，3] |
| 5，7 进入 Primes 集合 | [ ] | [2，3，5，7] |

表 3．4．1

然后搜索 primes 集合中的每一个素数 i，分别作为等差数列的第 1 个元素，计算以素

数 i 后的每一个元素与素数 i 的间距为公差的数列。显然，按照上述算法，可以得出所有可能的等差数列。其中长度最长的等差数列即为问题的解。

```
var
 s,primes:set of 1..255;                                  {s—筛；primes—素数集合}
 f,f1,ff,c:string;
        {f-数列；f1-数列的当前元素；f-f 数列的第一个元素；c-长度最长的等差数列}
 d,p,n,i,j,delta,k,max:integer;
          {P—筛中最小数；d—P 的倍数，应从筛中去掉；delta—公差；max—最大长度}
begin
 readln(n);                                                          {读问题规模}
 s:=[2..n]; p:=1; primes:=[];                        {筛、筛中最小数和素数集合初始化}
 repeat
  repeat p:=p+1;  until p in s;                              {选取筛中最小数为素数}
  primes:=primes+[p];                                                {P 放入素数表}
  d:=p;
  repeat
   s:=s-[d];d:=d+p;                                          {去掉筛中所有 P 的倍数}
  until d>n;
 until s=[];                                   {直到筛空，即 2..N 上的所有素数求出为止}
 for i:=2 to n do                                          {搜索素数集合中的每一个素数}
  if i in primes
   then begin
        str(i,ff); f:=ff;       {将素数 i 转换成数串 ff,作为等差数列的第一个元素}
        for j:=i+1 to n do                                  {顺序搜索 i 后的每一个素数}
         if j in primes
          then begin
               delta:=j-i;  k:=delta;                                       {计算公差}
               while(i+k<=n)and((i+k) in primes)do
                begin  {将公差为 delta 的素数（i+k，对应的数串为 f1）送入等差数列}
                 str(i+k,f1); f:=f+ ’  ’ +f1; k:=k+delta;
                end; {while}
               if length(f)>max                       {若等差数列的长度最长，则记下}
                  then begin max:=length(f);  c:=f;  end; {then}
               f:=ff;                          {继续搜索由素数 i 开始的、公差更大的数列}
               end; {then}
         end;
  writeln(’ The max length ’ ,max);                                   {输出最佳长度}
  writeln(c);                                                {输出最佳的等差数列}
 end.{main}
```

# 【上机练习】

## 1、高度与宽度(文件名：aa.pas)

输入杂乱排列的N个正整数，求出所有的最大平台（即出现次数最多的数）的平台宽度（最大平台数的个数）和平台高度（即最大平台中数的值）。例如：

**输入：(aa.in)**

10　　（正整数的个数N，N ≤ 30）

1 2 3 1 2 3 1 2 1 2　（N个正整数，每两个间空一格）

**输出：(aa.out)**

1: its width is 4,its height is 1.

2: its width is 4,its height is 2.

## 2、求级数和（SERIES.PAS）

**问题描述：**

我们让计算机来做一道数学题：计算 S = 1/1!+1/2!+1/3!+...+1/ N!

(其中 N! = 1*2*3*...* N)

键盘输入整数 N (1≤N≤1000)。屏幕输出 S，四舍五入到 15 位小数。

**输入输出样例：**

INPUT：N = 3

OUTPUT：S = 1.666666666666667

## 3、密码破译(PASSWORD.PAS)

**问题描述：**

某组织欲破获一个外星人的密码，密码由一定长度的字串组成。此组织拥有一些破译此密码的长度不同的钥匙，若两个钥匙的长度之和恰好为此密码的长度，则此密码被成功破译。现在就请你编程找出能破译此密码的两个钥匙。

**输入文件（PASSWORD.IN）：**

输入文件第一行为钥匙的个数N（1≤N≤5000）

输入文件第二行为密码的长度

以下N行为每个钥匙的长度

**输出文件（PASSWORD.OUT）：**

若无法找到破译此密码的钥匙，则输出文件仅1行0 。

若找到两把破译的钥匙，则输出文件有两行，分别为两把钥匙的编号。

若有多种破译方案，则只输出一种即可。

**输入输出样例：**

PASSWORD.IN

10

80

```
27
9
4
73
23
68
12
64
92
24

PASSWORD.OUT
6
7
```

**题目要求：**

所有长度、计算均在 Longint 范围之内。程序运行限定时间：2 秒

## 4、彩票摇奖(LOTTERY.PAS)

**问题描述：**

为了丰富人民群众的生活、支持某些社会公益事业，北塔市设置了一项彩票。该彩票的规则是：

(1) 每张彩票上印有7 个各不相同的号码，且这些号码的取指范围为1~33。

(2) 每次在兑奖前都会公布一个由七个各不相同的号码构成的中奖号码。

(3) 共设置7 个奖项，特等奖和一等奖至六等奖。兑奖规则如下：

特等奖：要求彩票上7 个号码都出现在中奖号码中。

一等奖：要求彩票上有6 个号码出现在中奖号码中。

二等奖：要求彩票上有5 个号码出现在中奖号码中。

三等奖：要求彩票上有4 个号码出现在中奖号码中。

四等奖：要求彩票上有3 个号码出现在中奖号码中。

五等奖：要求彩票上有2 个号码出现在中奖号码中。

六等奖：要求彩票上有1 个号码出现在中奖号码中。

注：兑奖时并不考虑彩票上的号码和中奖号码中的各个号码出现的位置。例如，中奖号码为23 31 1 14 19 17 18，则彩票12 8 9 23 1 16 7 由于其中有两个号码(23 和1)出现在中奖号码中，所以该彩票中了五等奖。

现已知中奖号码和小明买的若干张彩票的号码，请你写一个程序帮助小明判断他买的彩票的中奖情况。

**输入文件(LOTTERY.IN)：**

输入文件的第一行只有一个自然数 N ≤ 1000，表示小明买的彩票张数；第二行存放了 7 个介于1 和33 之间的自然数，表示中奖号码；在随后的 N 行中每行都有 7 个介于 1 和 33 之间的自然数，分别表示小明所买的 N 张彩票。

**输出文件(LOTTERY.OUT)：**

依次输出小明所买的彩票的中奖情况(中奖的张数)，首先输出特等奖的中奖张数，然后依次输出一等奖至六等奖的中奖张数。

**输入输出样例：**

LOTTERY.IN
2
23 31 1 14 19 17 18
12 8 9 23 1 16 7
11 7 10 21 2 9 31

LOTTERY.OUT
0 0 0 0 0 1 1

**5、周期串（PERIODIC.PAS）**

**问题描述：**

如果一个字符串是以一个或者一个以上的长度为 k 的重复字符串所连接成的，那么这个字符串就被称为周期为 k 的字符串。例如，字符串”abcabcabcabc”周期为 3，因为它是由 4 个循环”abc”组成的。它同样是以 6 为周期（两个重复的”abcabc”）和以 12 为周期（一个循环”abcabcabcabc”）。

写一个程序，读入一个字符串，并测定它的最小周期。

**输入文件（PERIODIC.IN）：**

一个最长为 100 的没有空格的字符串。

**输出文件（PERIODIC.OUT）：**

一个整数表示输入的字符串的最小周期。

**输入输出样例：**

PERIODIC.IN
HoHoHo

PERIODIC.OUT
2

## 6、生日日期(BIRTHDAY.PAS)

**问题描述：**

小甜甜的生日是 YY 年 MM 月 DD 日，他想知道自己出生后第一万天纪念日的日期（出生日算第 0 天）。

## 输入文件（BIRTHDAY.IN）：

从文件的第一行分别读入 YY，MM，DD 其中 1949 <= YY <= 2002，日期绝对合法。

**输出文件（BIRTHDAY.OUT）：**

输出文件只有一行，即小甜甜生日第一万天以后的日期，格式为 “YY-MM-DD”。

**输入输出样例：**

| BIRTHDAY.IN | BIRTHDAY.OUT |
|---|---|
| 1975 7 15 | 2002-11-30 |

# 第二章　枚举法

所谓枚举法,指的是从可能的解集合中一一枚举各元素,用题目给定的检验条件判定哪些是无用的,哪些是有用的.能使命题成立,即为其解。一般思路：

①对命题建立正确的数学模型；

②根据命题确定的数学模型中各变量的变化范围（即可能解的范围）；

③利用循环语句、条件判断语句逐步求解或证明；

枚举法的特点是算法简单，但有时运算量大。对于可能确定解的值域又一时找不到其他更好的算法时可以采用枚举法。

**【例 1】**求满足表达式 A+B=C 的所有整数解，其中 A，B，C 为 1~3 之间的整数。

分析：本题非常简单，即枚举所有情况，符合表达式即可。算法如下：

```
for A := 1 to 3 do
   for B := 1 to 3 do
     for C := 1 to 3 do
       if A + B = C then
         Writeln(A, '+', B, '=', C);
```

上例采用的就是枚举法。所谓枚举法，指的是从可能的解的集合中一一枚举各元素，用题目给定的检验条件判定哪些是无用的，哪些是有用的。能使命题成立的，即为解。

从枚举法的定义可以看出，枚举法本质上属于搜索。但与隐式图的搜索有所区别，在采用枚举法求解的问题时，必须满足两个条件：

预先确定解的个数 n；

对每个解变量 A1，A2，……，An 的取值，其变化范围需预先确定

$A_1 \in \{X_{11}, \cdots\cdots, X_{1p}\}$

……

$A_i \in \{X_{i1}, \cdots\cdots, X_{iq}\}$

……

$A_n \in \{X_{n1}, \cdots\cdots, X_{nk}\}$

**【例 1】**中的解变量有 3 个：A，B，C。其中

A 解变量值的可能取值范围 A∈{1，2，3}

B 解变量值的可能取值范围 B∈{1，2，3}

C 解变量值的可能取值范围 C∈{1，2，3}

则问题的可能解有 27 个

（A，B，C）∈{（1，1，1），（1，1，2），（1，1，3），
（1，2，1），（1，2，2），（1，2，3），
……
（3，3，1），（3，3，2），（3，3，3）}

在上述可能解集合中，满足题目给定的检验条件的解元素，即为问题的解。

如果我们无法预先确定解的个数或各解的值域，则不能用枚举，只能采用搜索等算法求解。由于回溯法在搜索每个可能解的枚举次数一般不止一次，因此，对于同样规模的问题，回溯算法要比枚举法时间复杂度稍高。

**【例 2】** 给定一个二元一次方程 aX+bY=c。从键盘输入 a,b,c 的数值，求 X 在[0，100]，Y 在[0，100]范围内的所有整数解。

分析：要求方程的在一个范围内的解，只要对这个范围内的所有整数点进行枚举，看这些点是否满足方程即可。参考程序：

```
program exam8;
var
   a,b,c:integer;
   x,y:integer;
begin
     write('Input a,b,c:');readln(a,b,c);
     for x:=0 to 100 do
        for y:=0 to 100 do
           if a*x+b*y=c then writeln(x,' ',y);
end.
```

从上例可以看出,所谓枚举法,指的是从可能的解集合中一一枚举各元素,用题目给定的检验条件判定哪些是无用的,哪些是有用的.能使命题成立,即为其解。

**【例 3】**找出 n 个自然数(1,2,3,…,n)中 r 个数的组合。例如，当 n= 5 , r=3 时，所有组合为：

```
5      4      3
5      4      2
5      4      1
5      3      2
5      3      1
5      2      1
4      3      2
4      3      1
4      2      1
3      2      1
total=10      {组合的总数}
```

**【算法分析】**

n 个数中 r 的组合，其中每 r 个数中，数不能相同。另外，任何两组组合的数，所包含的数也不应相同。例如，5、4、3 与 3、4、5。为此，约定前一个数应大于后一个数。将上述两条不允许为条件，当 r=3 时，可用三重循环进行搜索。

**【参考程序】**

```
Program  zuhe11;
  const n=5;
  var  i,j,k,t:integer;
  begin  t:=0;
    for  i:=n  downto  1  do
      for  j:=n  downto  1  do
        for  k:=n  downto  1  do
          if (i>j)and(j>k) then
            begin t:=t+1;writeln(i:3,j:3,k:3);end;
    writeln('total=',t);
  end.
```

或者

```
Program  zuhe12;
const n=5;r=3;
var  i,j,k,t:integer;
begin  t:=0;
  for  i:=n  downto  r  do
    for  j:=i-1  downto  r-1  do
      for  k:=j-1  downto  1  do
          begin t:=t+1;writeln(i:3,j:3,k:3);end;
  writeln('total=',t);
end.
```

这两个程序，前者枚举了所有可能情形，从中选出符合条件的解，而后者比较简洁。但是这两个程序都有一个问题，当 r 变化时，循环重数改变，这就影响了这一问题的解，即没有一般性。

但是，很多情况下枚举搜索法还是常用的。

**【例 4】巧妙填数(NOIP1998)**

将 1～9 这九个数字填入九个空格中。每一横行的三个数字组成一个三位数。如果要使第二行的三位数是第一行的两倍，第三行的三位数是第一行的三倍，应怎样填数。如图 6:

| | | |
|---|---|---|
| 1 | 9 | 2 |
| 3 | 8 | 4 |
| 5 | 7 | 6 |

图 6

**【算法分析】**

本题目有 9 个格子，要求填数，如果不考虑问题给出的条件，共有 9！=362880 种方案，在这些方案中符合问题条件的即为解。因此可以采用枚举法。

但仔细分析问题，显然第一行的数不会超过 400，实际上只要确定第一行的数就可以根据条件算出其他两行的数了。这样仅需枚举 400 次。因此设计参考程序：

**解法一:**

```
program exam9;
var   i,j,k,s:integer;
function sum(s:integer):integer;
begin
  sum:=s div 100 + s div 10 mod 10 + s mod 10
end;
function mul(s:integer):longint;
begin
  mul:=(s div 100) * (s div 10 mod 10) * (s mod 10)
end;
begin
   for i:=1 to 3 do
   for j:=1 to 9 do
```

```
  if j<>i then for k:=1 to 9 do
  if (k<>j) and (k<>i) then begin
    s := i*100 + j*10 +k;        {求第一行数}
    if 3*s<1000 then
         if (sum(s)+sum(2*s)+sum(3*s)=45) and
           (mul(s)*mul(2*s)*mul(3*s)=362880) then {满足条件,并数字都由 1~9 组成}
      begin
         writeln(s);
         writeln(2*s);
         writeln(3*s);
         writeln;
      end;
  end;
end.
```

**解法二:**

```
    program sheep308;
    var s:string[9];
        i:integer;
        x1,y1,z1:string[3];
        k:char;

     function check(x,y,z:integer):boolean;
      begin
       str(x,x1);str(y,y1);str(z,z1);
       s:=x1+y1+z1;
       check:=true;
       for k:='1' to '9' do
        if pos(k,s)=0 then check:=false;
      end;
     begin
      for i:=123 to 329 do
        if check(i,i*2,i*3) then writeln(i:5,i*2:5,i*3:5);
      readln;
 end.
```

**解法三:**

```
    program d5;
    var i,x,y,z:integer;
        a:set of 1..9;
    begin
      for i:=100 to 333 do
      begin
        a:=[1..9];
        x:=i; y:=2*i;   z:=3*i;
        repeat
          a:=a-[x mod 10]-[y mod 10]-[z mod 10];
```

```
        x:=x div 10; y:=y div 10; z:=z div 10;
      until (a=[]) or (x=0);
     if a=[] then writeln(i:4,2*i:4,3*i:4);
    end;
    readln;
end.
```

**【例 5】**在某次数学竞赛中，A、B、C、D、E 五名学生被取为前五名。请据下列说法判断出他们的具体名次，即谁是第几名？

条件 1：你如果认为 A，B，C，D，E 就是这些人的第一至第五名的名次排列，便大错。因为：

没猜对任何一个优胜者的名次。

也没猜对任何一对名次相邻的学生。

条件 2：你如果按 D，A，E，C，B 来排列五人名次的话，其结果是：

说对了其中两个人的名次。

还猜中了两对名次相邻的学生的名次顺序。

**【算法分析】**本题是一个逻辑判断题，一般的逻辑判断题都采用枚举法进行解决。5 个人的名次分别可以有 5!=120 种排列可能，因为 120 比较小，因此我们对每种情况进行枚举，然后根据条件判断哪些符合问题的要求。

根据已知条件，A<>1,B<>2,C<>3,D<>4,E<>5,因此排除了一种可能性，只有 4！=24 种情况了。

**【参考程序】**

```
Program Exam10;
Var  A,B,C,D,E              :Integer;
     Cr                     :Array[1..5] Of Char;
Begin
  For A:=1 To 5 Do
    For B:=1 To 5 Do
     For C:=1 To 5 Do
      For D:=1 To 5 Do
       For E:=1 To 5 Do Begin
        {ABCDE 没猜对一个人的名次}
          If (A=1) Or (B=2) Or (C=3) Or (D=4) Or (E=5) Then Continue;
          If [A,B,C,D,E]<>[1,2,3,4,5] Then Continue;{他们名次互不重复}
          {DAECB 猜对了两个人的名次}
          If Ord(A=2)+Ord(B=5)+Ord(C=4)+Ord(D=1)+Ord(E=3)<>2 Then Continue;
        {ABCDE 没猜对一对相邻名次}
        If (B=A+1) Or (C=B+1) Or (D=C+1) Or (E=D+1) Then Continue;
        {DAECB 猜对了两对相邻人名次}
        If Ord(A=D+1)+Ord(E=A+1)+Ord(C=E+1)+Ord(B=C+1)<>2 Then Continue;
          Cr[A]:='A';Cr[B]:='B';Cr[C]:='C';
          Cr[D]:='D';Cr[E]:='E';
          WRITELN(CR[1],' ',CR[2],' ',CR[3],' ',CR[4],' ',CR[5]);
        End;
End.
```

【例 6】有长度为 3、4、5、6、7、8、9 的小木棍各一根，从中选取三根作为三角形的边长，问能构成多少种不同类型的三角形，编程打印输出各种不同类型的三角形。

```
Var   i,j,k,n : byte;
begin
  n := 0;
  for i := 3 to 7 do
    for j := i+1 to 8 do
      for k := j+1 to 9 do
        if (i+j > k) and (i+k > j) and (j+k > i) then begin
          n := n+1;
          writeln('No.',n:2,' :',i:5,j:5,k:5);
          if n mod 24 = 0 then readln;
        end;
  readln;
end.
```

## 【上机练习】

**1**、警察局抓了 a、b、c、d 四名偷窃嫌疑人，其中有一人是小偷。审问中：

a 说："我不是小偷。"　　b 说："c 是小偷。"

C 说："小偷肯定是 d。"　　d 说："c 是冤枉人!"

现在已经知道四人中三人说的是真话，一人说的假话。问到底谁是小偷。P19

**2**、甲乙丙丁戊五个人在运动会上分获百米、二百米、跳高、跳远和铅球冠军，有四个猜测比赛结果：

a 说：乙获铅球冠军，丁获跳高冠军。　　b 说：甲获百米冠军，戊获跳远冠军。

C 说：丙获跳远冠军，丁获二百米冠军。　　d 说：乙获跳高冠军，戊获铅球冠军。

其中每个人都只说对一句，说错一句。求五人各获哪项冠军。P19

**3**、古希腊人认为因子的和等于它本身的数是一个完全数（自身因子除外）。例如 28 的因子是 1，2，4，7，14，且 1+2+4+7+14=28，则 28 是一个完全数。编写一个程序 2—1000 内的所有完全数。

**4**、邮局发行一套票面有四种不同值的邮票，如果每封信所贴邮票张数不超过三枚，存在整数 R，使得用不超过三枚的邮票，可以帖出连续的整数 1、2、3,…,R 来，找出这四种面值数，使得 R 值最大。（名师奥赛 P270）

**5**、有 A、B、C、D、E 五本书，要分给张、王、刘、赵、钱五位同学，每人只能选一本，事先让每人把自己喜爱的书填于下表，编程找出让每人都满意的所有方案。

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 张 | | | Y | Y | |
| 王 | Y | Y | | | Y |
| 刘 | | Y | Y | | |
| 赵 | Y | Y | | Y | |
| 钱 | | Y | | | Y |

【答案】四种方案

| | 张 | 王 | 刘 | 赵 | 钱 | | 张 | 王 | 刘 | 赵 | 钱 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ① | C | A | B | D | E | ② | D | A | C | B | E |
| ③ | D | B | C | A | E | ④ | D | E | C | A | B |

# 第三章　不同进制数的转换及应用

在计算机领域中,有时需要将十进制数转换成二进制、八进制和十六进制的数，有时又需要逆向转换将二进制、八进制和十六进制的数转换成十进制或它们相互间进行转换。

不同进制转换的基本算法是：

（1） 十进制数(x)转换成任意进制数(n)的方法:将十进制数除以 n 进制反序取余.

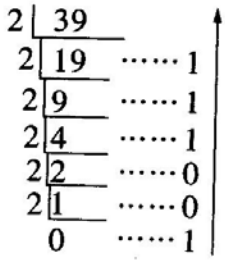
（2） 将任意进制数转换成十进制数方法:按”权”展开.

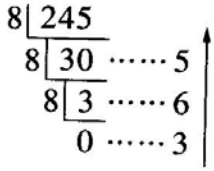
（3） 二进制、八进制、十六进制之间的转换方法：利用 3 位二进制表示 1 位八进制数，4 位二进制数可以表示 1 位十六进制数的方法。

**1、不同进制数的转换**

（1） 十进制整数转换成任意进制，除以进制反向取余。

短除法$(39)_{10}$=（$100111$）$_2$　　$(245)_{10}=(365)_8$




设任意进制为 x,十进制数为 y 则其算法模式是:

重复做:

t:=t+1;

y mod x 的余数:a(t)

y:=y div x

直到 y=0 为止.

输出则从 a(t)最高位到第一位 a(1).

(2)任意进制整数转换成十进制,按权展开

设任意进制 n 数为 x,按权展开的模式是:

$(1101)_2=1\times2^3+1\times2^2+0\times2^1+1\times2^0=13$

$(165)_8=1\times8^2+6\times8^1+5\times8^0=117$

对应这两种转换程序如下:

**【例 1】将十进制整数转换成任意进制的数(设进制 n<10)**

```
program p12-5;
  var
      a:array[1..50] of integer;
      n,x,d,i:integer;
  begin
    write( ‘input number x,d:’ );
    read(x,d);i:=0;
```

```
    writeln;
    repeat
      i:=i+1;
      a[i]:=x mod d;
      x:=x div d;
    until x=0;
    for n:=I downto 1 do
        write(a[n]);
    writeln;
end.
```

当程序程序运行后:

显示:input number x,y:

输入:245 8

输出:365

如果超过十进制,可以用字符A,B,C,D,E,F表示10~15的数,将余数再转换为字符类型进行处理,有关这部分程序,请读者自己完成.

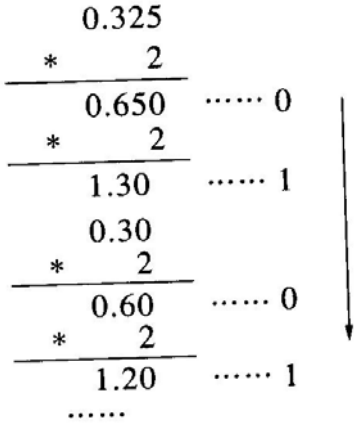
**【例 2】 将任意进制整数转换成十进制数**

```
program p12-6;
  const m=20;
  var str1:string; str2:char;
      n,I,l,y,t:integer; x:longint;
      a:array[1..m] of integer;
  begin
    writeln( ‘input number x,n:’ );
    readln(x,n);
    str(x,str1); l:=length(str1);
    for i:=1 to l do
      begin
        str2:=str1[I];
        a[i]:=ord(str2)-ord( ‘0’ );
      end;
    y:=1; t:=a[L];
    for i:=l-1 downto 1 do
      begin
        if a[i]>n then
          begin
            writeln( ‘error’ );
            exit;
          end;
        y:=y*n;
        t:=t+a[i]*y;
      end;
      writeln(x,’ →’ t);
  end.
```

当 x=100110,n=2,输出结果 100110→38

当 x=46,n=8,输出结果 46→38

(3)将十进制小数小数转换为其它进制的数,其基本算法是:将小数乘以待转换的进制数正向取整的方法.例如(0.325)10=(　　)2,其计算方法如下计算式:




所以,运算结果是(0.325)10≈(0.0101)2,

(0.325)10≈(0.2463)8,其转换算法的程序,读者自己完成.

**2.数的进制的应用**

在解决实际问题时,巧妙应用数的进制的特点,可以使问题简化.

**【例 3】用三进制数求解数学问题.**

用质量为 1g,3g,9g,27g 和 81g 的砝码称物体的质量,最大可称 121g.如果砝码允许放在天平的两边,编程输出称不同物体时砝码应该怎样安排?例如 m=14 时,m+9+3+1=27 或 m=27-9-3-1.即天平一端放 m=14 克的物体和 9g,3g,1g 的砝码,另一端放 27g 的砝码.

**【算法分析】**

(1) 被称物质的质量计算的数学原理:设被称物质 m 放在天平左边,根据天平平衡原理,左边质量应等于天平右边质量.

(2) 问题关键在于算法中如何体现砝码放在天平左边,右边参加称量.这里可以用 -1,1,0 表示砝码放在天平左右和没有参加称量,再没有其他数,所以称为三进制数,每个砝码都有这样的三种状态.

(3) 被称物体质量计算:m=a*81+b*27+c*9+d*3+e

这里 a,b,c,d,e 分别表示 81,27,9,3,1 克的砝码是放在天平的左边,右边,其程序表示如下:

```
program p12-7;
  var
      a,b,c,d,e,m:integer;
  begin
    for m:=1 to 121 do
      for a:=0 to 1 do
        for b:=-1 to 1 do
          for c:=-1 to 1 do
            for d:=-1 to 1 do
```

```
            for e:=-1 to 1 do
              if m=a*81+b*27+c*9+d*3+e then
                begin
                writeln(m,' =' ,a*81,' +' ,b*27,' +' ,c*9,' +' ,d*3,' +' ,e);
                end;
  end.
```

运行结果是:

1=0+0+0+0+1

……..

31=0+27+0+3+1

……

121=81+27+9+3+1

**【例 4】 走路问题**

小明每天上学要从街口 A 到街口 B,求他从 A 到 B 的向前路(不后退)一共有多少种走法?应该怎样走?(如下图)

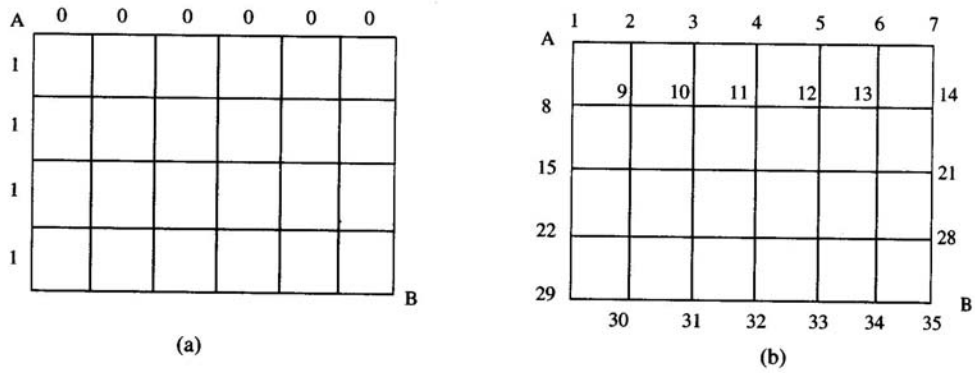



图 12－1

**问题分析:**

(1) 我们可以利用二进制数来表示从 A 到 B 的走法:设向右走记为 0;向下走记为 1;

(2) 用 m 表示行数,即横向街道数:n 表示列数,即纵向街道数.

(3) 从 A 到 B 一共走 10 步,且步数的和为 10,我们用 x,I,j,k,a 这五种循环分别记下每走一步的情况(用 1 或 0 表示).

**【参考程序】**

```
program p12-8;
  const m=4;n=6;
  var t,p,x,I,j,k,a:integer;
  begin
    t:=1;
    p:=m+n;
    for x:=1 to p-(m-1) do
      for i:=x+1 to p-(m-2) do
```

```
        for j:=x+1 to p-(m-3) do
          for k:=j+1 to p do
            begin
              write( 'no:' ,t,' ---- ');
              for a:=1 to p do
                if (a=x) or (a=i) or (a=j) or (a=k) then write(1:2) else write(0:2);
              writeln:
              t:=t+1;
            end;
    writeln( 'total=' ,t-1);
  end.
```

当程序运行后,其结果如下列形式:

no: 1-1 1 1 1 0 0 0 0 0 0

no: 2-1 1 0 1 0 0 0 0 0 0

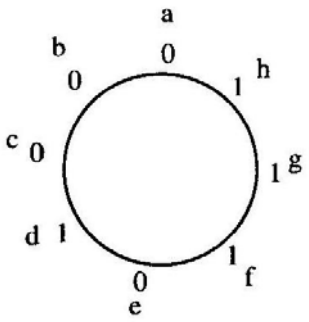
……

no: 210 0 0 0 0 0 0 1 1 1 1

total=210

此题的另一种解法,用图 12-1(b)标码,再用五重循环语句完成.读者可自行完成.

**【例 5】**将 $2^n$ 个 0 和 $2^n$ 个 1 排成一圈.从任意一个位置开始,每次按逆时针的方向以长度为 n+1 的单位计数二进制数.要求给出一种排法,用上面的方法产生出 2n+1 个不相同的二进制数.如当 n=2 时,有 $2^2$ 个 0 和 $2^2$ 个 1 排列如图所示.如果从 a 位置开始,逆时针方向取三个数 000,然后再从 b 位置上开始取三个数 001,接着取 010,…..,可以得到 8 个不同的二进制数.




**问题分析:**

(1) 若要产生 n 位二进制数,则有 $2^{n-1}$ 个 0 和有 $2^{n-1}$ 个 1,由这些 0,1 组成 n 位的二进制数,所以用数组记录 $2^{n-1}$ 个 0 和有 $2^{n-1}$ 个 1 的排列方法.

(2) 若要产生三位二进制数,即 n=3 时,则有

a[1]=0,a[2]=0,…,a[8]=1,表示为:

a1a2a3=0, a2a3a4=1, a3a4a5=2,…,a7a8a1=6,a8a1a2=4,数组需多定义 2 位.为此,一般情况多定义 n-1 位.

(3) 所产生的不同二进制数的判断方法:将产生的二进制转换成十进制数,其范围是 0~2n-1 的整数,判断所产生的二进制数是否在此范围内;

(4) 从后向前产生数据 t. t=ai-1*2n-1+t/2,若 t 在 s 中可以接受,否则不可以接受,另换一个值.

**【参考程序】**(设 n=4)

```
program p12-9;
  const maxn=16;m=127;
  type ts=set of 0..m;
  var n:1..maxn;
      a:array[1..m] of integer;
      s:ts;
      p,y,I,:integer;
  function bbb(k,t:integer):boolean;
```

```
    var b:boolean;
    begin
      if k=0 then bbb:=true      {k 表示位数}
        else begin
                t:=t div 2;      {产生一个数}
                if t in s then
                  begin
                    s:=s-[t];    {从集合 s 中除去 t}
                    a[k]:=0;
                    b:=bbb(k-1,t);
                    if not b then s:=s+[t];
                  end;
                else b:=false;
                if not b then
                  begin
                    t:=t+p div 2;          {产生一个数}
                    if t in s then begin
                                     s:=s-[t];
                                     a[k]:=1;      {n 个 1}
                                     b:=bbb(k-1,t);
                                     if not b then s:=s+[t]
                                   end;
                                else b:=false;
                  end;
                bbb:=b;
            end;
    end;

begin                    {主程序}
  readln(n);
  p:=1;
  for i:=1 to n do       {计算 2^n}
      p:=p*2;
  for i:=1 to p+n-1 do   {数驵 a 初始化}
      a[i]:=0;
  s:=[1..p-1];
  if bbb(p-1,0) then     {调用函数}
        for i:=1 to p do write(a[i]:2)  {输出每一位}
    else write( 'no solution' );
    writeln;
end.
```

程序程序运行结果是:

输入 n: 4

输出结果:

0 0 0 1 1 1 1 0 1 0 1 1 0 0 1 0

# 第四章　高精度计算

利用计算机进行数值计算,有时会遇到这样的问题:有些计算要求精度高,希望计算的数的位数可达几十位甚至几百位,虽然计算机的计算精度也算较高了,但因受到硬件的限制,往往达不到实际问题所要求的精度.我们可以利用程序设计的方法去实现这样的高精度计算.这里仅介绍常用的几种高精度计算的方法.

高精度计算中需要处理好以下几个问题:

（1）数据的接收方法和存贮方法

数据的接收和存贮:当输入的数很长时,可采用字符串方式输入,这样可输入数字很长的数,利用字符串函数和操作运算,将每一位数取出,存入数组中.另一种方法是直接用循环加数组方法输入数据.

(2) 计算结果位数的确定

位数的确定:利用对数函数

l=trunc(ln(x)/ln(10))+1,定义数组 a[i];

(3) 进位,借位处理

**加法进位**:C[i]:= A[i]+B[i];

if C[i]>=10 then begin C[i]:= C[i] mod 10; C[i+1]:= C[i+1]+1 end;

**减法借位**:if a[i]<b[i] then begin a[i+1]:=a[i+1]-1;a[i]:=a[i]+10 end

c[i]:=a[i]-b[i]

**乘法进位**:x:= A[i]*B[j]+ x DIV 10+ C[i+j-1];

C[i+j-1] := x mod 10;

(4) 商和余数的求法

商和余数处理:视被除数和除数的位数情况进行处理.

**例 1 从键盘读入两个正整数，求它们的和。**

**分析：**从键盘读入两个数到两个变量中，然后用赋值语句求它们的和，输出。但是，我们知道，在 pascal 语言中任何数据类型都有一定的表示范围。而当两个被加数据大时，上述算法显然不能求出精确解，因此我们需要寻求另外一种方法。在读小学时，我们做加法都采用竖式方法，如图 1。 这样，我们方便写出两个整数相加的算法。

$$\begin{array}{r} 856 \\ +\ \ 255 \\ \hline 1111 \end{array} \qquad\qquad \begin{array}{r} A_3A_2A_1 \\ +\ \ B_3B_2B_1 \\ \hline C_4C_3C_2C_1 \end{array}$$

图 1　　图 2

如果我们用数组 A、B 分别存储加数和被加数，用数组 C 存储结果。则上例有 A[1]=6, A[2]=5, A[3]=8, B[1]=5,B[2]=5, B[3]=2, C[4]=1,C[3]=1, C[2]=1,C[1]=1,两数相加如图 2 所示。

**因此，算法描述如下：**

```
procedure add(a,b;var c);
{ a,b,c 都为数组，a 存储被加数，b 存储加数，c 存储结果 }
var i,x:integer;
```

```
begin
   i:=1
   while (i<=a 数组长度) or(i<=b 数组的长度) do begin
       x := a[i] + b[i] + x div 10;  {第 i 位相加并加上次的进位}
       c[i] := x mod 10;             {存储第 i 位的值}
       i := i + 1                    {位置指针变量}
   end
end;
```

通常，读入的两个整数用可用字符串来存储，**程序设计如下：**

```
program exam1;
  const  max=200;
  var   a,b,c:array[1..max] of 0..9;
        n:string;
        lena,lenb,lenc,i,x:integer;
begin
     write('Input augend:'); readln(n);
     lena:=length(n);                      {加数放入 a 数组}
     for i:=1 to lena do a[lena-i+1]:=ord(n[i])-ord('0');
     write('Input addend:'); readln(n);
     lenb:=length(n);                      {被加数放入 b 数组}
     for i:=1 to lenb do b[lenb-i+1]:=ord(n[i])-ord('0');
     i:=1;
     while (i<=lena) or(i<=lenb) do
      begin
         x := a[i] + b[i] + x div 10;      {两数相加，然后加前次进位}
         c[i] := x mod 10;                 {保存第 i 位的值}
         i := i + 1
     end;
    if x>=10 then                           {处理最高进位}
      begin lenc:=i;c[i]:=1; end
    else lenc:=i-1;
    for i:=lenc downto 1 do write(c[i]);   {输出结果}
    writeln
end.
```

**例 2 高精度减法。从键盘读入两个正整数，求它们的差。**

**分析：**类似加法，可以用竖式求减法。在做减法运算时，需要注意的是：被减数必须比减数大，同时需要处理借位。

**高精度减法的参考程序：**

```
program exam2;
const
     max=200;
```

```
var
   a,b,c:array[1..max] of 0..9;
   n,n1,n2:string;
   lena,lenb,lenc,i,x:integer;
begin
     write('Input minuend:'); readln(n1);
     write('Input subtrahend:'); readln(n2);        {处理被减数和减数}
     if (length(n1)<length(n2)) or (length(n1)=length(n2)) and (n1<n2) then
        begin
          n:=n1;n1:=n2;n2:=n;
          write('-')                                {n1<n2,结果为负数}
        end;
     lena:=length(n1); lenb:=length(n2);
     for i:=1 to lena do a[lena-i+1]:=ord(n1[i])-ord('0');
     for i:=1 to lenb do b[lenb-i+1]:=ord(n2[i])-ord('0');
     i:=1;
     while (i<=lena) or(i<=lenb) do
       begin
         x := a[i] - b[i] + 10 + x;      {不考虑大小问题，先往高位借 10}
         c[i] := x mod 10 ;              {保存第 i 位的值}
         x := x div 10 - 1;              {将高位借掉的 1 减去}
         i := i + 1
      end;
    lenc:=i;
    while (c[lenc]=0) and (lenc>1) do dec(lenc);        {最高位的 0 不输出}
    for i:=lenc downto 1 do write(c[i]);
    writeln
end.
```

**例 3 高精度乘法。从键盘读入两个正整数，求它们的积。**

**分析：**类似加法，可以用竖式求乘法。在做乘法运算时，同样也有进位，同时对每一位进乘法运算时，必须进行错位相加，如图 3, 图 4。

分析 C 数组下标的变化规律，可以写出如下关系式：$C_i = C'_i + C''_i + \cdots$

由此可见，$C_i$跟 A[i]*B[j]乘积有关，跟上次的进位有关，还跟原 $C_i$的值有关，分析下标规律，有 x:= A[i]*B[j]+ x DIV 10+ C[i+j-1];

C[i+j-1] := x mod 10;

$$\begin{array}{r} 856 \\ \times\quad 25 \\ \hline 4280 \\ 1712\phantom{0} \\ \hline 21400 \end{array}$$

图 3

$$\begin{array}{r} A_3A_2A_1 \\ \times\quad B_3B_2B_1 \\ \hline C'_4C'_3C'_2C'_1 \\ C''_5C''_4C''_3C''_2\phantom{C''_1} \\ \hline C_6C_5C_4C_3C_2C_1 \end{array}$$

图 4

类似，高精度乘法的参考程序：

```
program exam3;
const
   max=200;
var
   a,b,c:array[1..max] of 0..9;
   n1,n2:string;
   lena,lenb,lenc,i,j,x:integer;
begin
     write('Input multiplier:'); readln(n1);
     write('Input multiplicand:'); readln(n2);
     lena:=length(n1); lenb:=length(n2);
     for i:=1 to lena do a[lena-i+1]:=ord(n1[i])-ord('0');
     for i:=1 to lenb do b[lenb-i+1]:=ord(n2[i])-ord('0');
     for i:=1 to lena do begin
       x:=0;
       for j:=1 to lenb do begin                {对乘数的每一位进行处理}
         x := a[i]*b[j] + x div 10 + c[i+j-1]; {当前乘积+上次乘积进位+原数}
         c[i+j-1] :=  x mod 10;
       end;
       c[i+j]:= x div 10;                       {进位}
     end;
     lenc:=i+j;
     while (c[lenc]=0) and (lenc>1) do dec(lenc);
     for i:=lenc downto 1 do write(c[i]);
     writeln
end.
```

**例 4　高精度除法。从键盘读入两个正整数，求它们的商（做整除）。**

**分析：**做除法时，每一次上商的值都在 0 ～ 9，每次求得的余数连接以后的若干位得到新的被除数，继续做除法。因此，在做高精度除法时，要涉及到乘法运算和减法运算，还有移位处理。当然，为了程序简洁，可以避免高精度乘法，用 0～9 次循环减法取代得到商的值。这里，我们讨论一下高精度数除以单精度数的结果，采取的方法是按位相除法。

```
program exam4;
```

```
const   max=200;
var  a,c:array[1..max] of 0..9;
     x,b:longint;  n1,n2:string;
     lena:integer; code,i,j:integer;
begin
     write('Input dividend:'); readln(n1);
     write('Input divisor:'); readln(n2);
     lena:=length(n1);
     for i:=1 to lena do  a[i] := ord(n1[i]) - ord('0');
     val(n2,b,code);
     x:=0;                                        {按位相除}
     for i:=1 to lena do begin
         c[i]:=(x*10+a[i]) div b;
         x:=(x*10+a[i]) mod b;
     end;
     j:=1;
     while (c[j]=0) and (j<lena) do inc(j);       {去除高位的 0}
     for i:=j to lena do write(c[i]) ;
     writeln
end.
```

实质上，在做两个高精度运算时候，存储高精度数的数组元素可以不仅仅只保留一个数字，而采取保留多位数（例如一个整型或长整型数据等），这样，在做运算（特别是乘法运算）时，可以减少很多操作次数。例如图 5 就是采用 4 位保存的除法运算，其他运算也类似。具体程序可以修改上述例题予以解决，程序请读者完成。

示例：123456789 ÷45 = 1’2345’6789 ÷ 45
= 274’3484
∵ 1 div 45 = 0 ， 1 mod 45=1
∴ 取 12345 div 45 = 274 ∵ 12345 mod 45 = 15
∴ 取 156789 div 45 = 3484
∴ 答案为 2743484，余数为 156789 mod 45 = 9

图 5

## 【上机练习】

**1**、求 N！的精确值(N 是一般整数)。
**2**、计算 A/B 的精确值，设 A，B 是一般整数，计算机可接受的数，精确小数后 20 位。
**3**、求 s=1+2+3+……+n 的精确值(n 是一般整数)。
**4**、将例 1-例 4 的例题中程序用过程函数完善程序。

# 第五章　数据排序

信息获取后通常需要进行处理,处理后的信息其目的是便于人们的应用.信息处理方法有多种,通常有数据的排序,查找,插入,删除,归并等操作.在前面的章节中,读者已经接触了一些这方面的知识,本节重点介绍数据排序的几种方法.

## 1. 选择排序

（1） 基本思想：每一趟从待排序的数据元素中选出最小（或最大）的一个元素，顺序放在已排好序的数列的最前，直到全部待排序的数据元素排完。

（2）排序过程：

【示例】：

初 始 关键字 [49 38 65 97 76 13 27 49]
第一趟排序后 13 [38 65 97 76 49 27 49]
第二趟排序后 13 27 [65 97 76 49 38 49]
第三趟排序后 13 27 38 [97 76 49 65 49]
第四趟排序后 13 27 38 49 [49 97 65 76]
第五趟排序后 13 27 38 49 49 [97 97 76]
第六趟排序后 13 27 38 49 49 76 [76 97]
第七趟排序后 13 27 38 49 49 76 76 [97]
最后排序结果 13 27 38 49 49 76 76  97

```
Procedure SelectSort(Var R : FileType); //对 R[1..N]进行直接选择排序 //
  Begin
    for I := 1 To N - 1 Do      //做 N - 1 趟选择排序//
      begin
        K := I;
        For J := I + 1 To N Do //在当前无序区 R[I..N]中选最小的元素 R[K]//
          Begin
            If R[J] < R[K] Then K := J
          end;
        If K <> I Then          //交换 R[I]和 R[K] //
          begin Temp := R[I]; R[I] := R[K]; R[K] := Temp; end;
      end;
End; //SelectSort //
```

## 2. 冒泡排序

**（1）、基本的冒泡排序**

①基本思想

依次比较相邻的两个数,把大的放前面,小的放后面.即首先比较第 1 个数和第 2 个数,大数放前,小数放后.然后比较第 2 个数和第 3 个数.....直到比较最后两个数.第一趟结束,最小的一定沉到最后.重复上过程,仍从第 1 个数开始,到最后第 2 个数.然后.....

由于在排序过程中总是大数往前,小数往后,相当气泡上升,所以叫冒泡排序.

下面是 6 个元素的排序的过程

```
4    5    7    1    2    3
└────┘
5    4    7    1    2    3
     └────┘
5    7    4    1    2    3
          └────┘
5    7    4    1    2    3
               └────┘
5    7    4    2    1    3
                    └────┘   第一趟结束
5    7    4    2    3    ①
└────┘
7    5    4    2    3    1
     └────┘
7    5    4    2    3    1
          └────┘
7    5    4    2    3    1
               └────┘       第二趟结束
7    5    4    3    ②    1
 └────┘
7    5    4    3    2    1
     └────┘
7    5    4    3    2    1
          └────┘            第三趟结束
7    5    4    ③    2    1
└────┘
7    5    4    3    2    1
     └────┘                 第四趟结束
7    5    ④    3    2    1
└────┘                      第五趟结束
⑦    ⑤    4    3    2    1
```

②算法实现

```
for i:=1 to 9 do
 for j:=1 to 10-i do
   if a[j]<a[j+1]
     then begin
            temp:=a[j];
            a[j]:=a[j+1];
            a[j+1]:=temp;
          end;
```

**（2）改进**

上例中,可以发现,第二趟结束已经排好序.但是计算机此时并不知道已经排好序.所以,还需进行一次比较,如果没有发生任何数据交换,则知道已经排好序,可以不干了.因此第三趟比较还需进行,第四趟、第五趟比较则不必要.

我们设置一个布尔变量 bo 来记录是否有进行交换.值为 false 进行了交换，true 则没有。

代码如下

```
i:=1;
repeat
  bo:=true;
  for j:=1 to 10-i
    if a[j]<a[j+1] then
        begin
          temp:=a[j];
          a[j]:=a[j+1];
          a[j+1]:=temp;
          bo:=false;
        end;
  inc(i);
until bo;
```

**（3）再次改进**

如果说是有 20 个元素.数据序列是 8,3,4,9,7 再后跟着 15 个大于 9 且已经排好序的数据。在第三趟后算法终止. 总共做了 19+18+17=54 次比较使得绝大多数已排好序的数据在一遍扫描后足以发现他们是排好序的情况下仍然被检查 3 遍。

我们改进如下

```
flag:=10;
while flag>0 do
  begin
    k:=flag-1;
    flag:=0;
    for i:=1 to k do
       if a[i]<a[i+1] then
                    begin
                      temp:=a[i];
                      a[i]:=a[i+1];
                      a[i+1]:=temp;
                      flag:=i;
                    end;
  end;
```

改进的冒泡算法对上述数据进行的比较次数是 19+4+2=24。

### 3. 插入排序

插入排序是一种简单的排序方法,其算法的基本思想是:

(1) 设 n 个数据已经按照顺序排列好(假定从小到大顺序,存放在 a 数组);

(2) 输入一个数据 x,将其放在恰当的位置,使整个数据序列仍然有序;

① 将 x 与 n 个数比较,while x>a[i] do i:=i+1;

② j:=I;

③ 将 a 数组的元素从 j 位置开始向后移动:for k:=n+1 downto j do a[k]:=a[k-1];

④ a[j]:=x;

(3) 输出已经插入完毕的有序数列.

例如:设 n=8,数组 a 中 8 个元素的数据:(36,25,48,12,65,43,20,580),执行插入排序程序后,其数据变动情况:

第 0 步:[36] 25 48 12 65 43 20 58

第 1 步:[25 36] 48 12 65 43 20 58

第 2 步:[25 36 48] 12 65 43 20 58

第 3 步:[12 25 36 48] 65 43 20 58

第 4 步:[12 25 36 48 65] 43 20 58

第 5 步:[12 25 36 43 48 65] 20 58

第 6 步:[12 20 25 36 43 48 65] 58

第 7 步:[12 20 25 36 43 48 58 65]

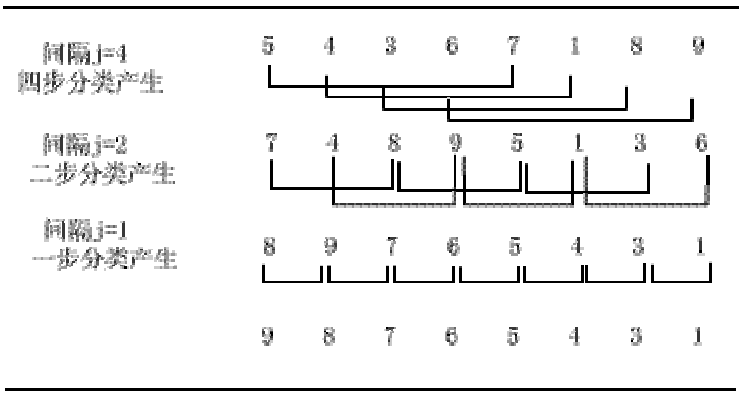
该算法的程序简单,读者自己完成.其算法的时间复杂性为 $O(n^2)$ 插入排序适用于原先数据已经排列好,插入一个新数据的情况.

```
Procedure InsertSort(Var R : FileType);
      //对 R[1..N]按递增序进行插入排序, R[0]是监视哨//
  Begin
    for I := 2 To N Do //依次插入 R[2],...,R[n]//
      begin
        R[0] := R[I]; J := I - 1;
        While R[0] < R[J] Do //查找 R[I]的插入位置//
          Begin
            R[J+1] := R[J]; //将大于 R[I]的元素后移//
            J := J - 1
          End
        R[J + 1] := R[0] ; //插入 R[I] //
      End
  End; //InsertSort //
```

### 4.希尔排序

**(1)基本思想**

希尔排序法是 1959 年由希尔(D.L.Shell)提出来的,又称减少增量的排序。下表是以八个元素排序示范的例子.在该例中,开始时相隔 4 个成分,分别按组进行排序,这时每组 2 个成分,共 4 组; 然后相隔 2 个成分,在按组排序.....最后,对所有相邻成分进行排序.




(2) 算法实现

```
j:=10;
i:=1;
while j>1 do
  begin
    j:=j div 2;
    repeat
      alldone:=true;
      for index:=1 to 10-j do
         begin
           i:=index+j;
           if a[index]<a[i] then
             begin
              temp:=a[index];
              a[index]:=a[i];
              a[i]:=temp;
              alldone:=false;
             end;
         end;
    until alldone
  end;
```

希尔排序中的问题是:究竟增量个数是多少,每次增量给定什么值(只知道逐次缩小,直到1),至今尚无十分明确的规则可以依据,只能凭经验来取.该算法是一种不稳定的排序,其时间复杂性约 $O(n^{1.3})$。**说句实话,这个很少有人用。**

5.快速排序

（1）基本思想：

在当前无序区 R[1..H]中任取一个数据元素作为比较的"基准"(不妨记为 X)，用此基准将当前无序区划分为左右两个较小的无序区：R[1..I-1]和 R[I+1..H]，且左边的无序子区

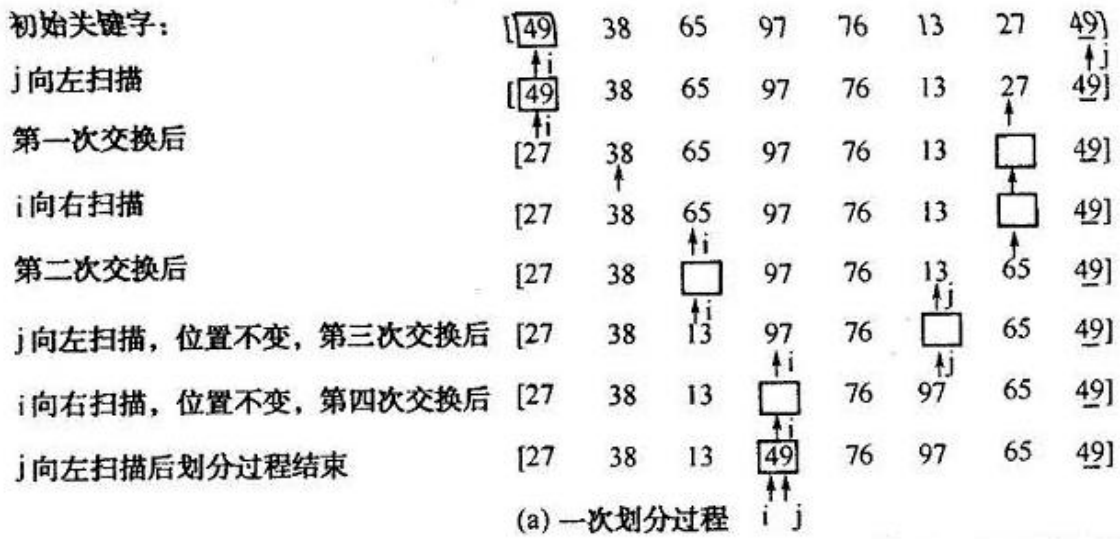
中数据元素均小于等于基准元素，右边的无序子区中数据元素均大于等于基准元素，而基准X则位于最终排序的位置上，即R[1..I-1]≤X.Key≤R[I+1..H](1≤I≤H)，当R[1..I-1]和R[I+1..H]均非空时，分别对它们进行上述的划分过程，直至所有无序子区中的数据元素均已排序为止。

**（2）排序过程：**

【示例】：

初 始 关键字　　[49 38 65 97 76 13 27 49]
第一次交换后　　[27 38 65 97 76 13 49 49]
第二次交换后　　[27 38 49 97 76 13 65 49]
J 向左扫描，位置不变，第三次交换后　[27 38 13 97 76 49 65 49]
I 向右扫描，位置不变，第四次交换后　[27 38 13 49 76 97 65 49]
J 向左扫描　　[27 38 13 49 76 97 65 49]

**（一次划分过程）**




初 始 关键字　[49　38　65　97　76　13　27　49]
一趟排序之后　[27　38　13] 49 [76　97　65　49]
二趟排序之后　[13] 27 [38] 49 [49　65] 76 [97]
三趟排序之后　13　27　38　49　49 [65] 76　97
最后的排序结果 13　27　38　49　49　65　76　97

**各趟排序之后的状态**

**快速排序算法 qsort**

```
Const      n=10;
Var    a:array [1..n] of integer;  k:integer;
```

```
procedure qsort(low,high:integer);
 var  i,j:integer;x:integer;
 begin
   if low<high then
     begin
       i:=low;j:=high;x:=a[i];
       repeat
         while (a[j]>=x)and(i<j)do j:=j-1;  {把大于基准的数留在右面}
         if (a[j]<x)and(i<j) then
           begin
             a[i]:=a[j];i:=i+1;            {把小于基准的数交换到左面}
           end;
         while (a[i]<=x)and (i<j) do i:=i+1;{把小于基准的数留在左面}
         if (a[i]>x)and(i<j) then
           begin
             a[j]:=a[i];j:=j-1;            {把大于基准的数交换到右面}
           end;
       until i=j;                  {直到 I 与 j 重叠，完成一次分组过程}
       a[i]:=x;            {把基准数插入相应的位置，以后不再参与排序}
       qsort(low,i-1);                          {小于基准的数重复分组}
       qsort(i+1,high);                         {大于基准的数重复分组}
     end;
   end;
```

**算法改进：**

```
procedure qsort(l,r:integer);
  var i,j,mid:integer;
  begin
    i:=l;j:=r; mid:=a[(l+r) div 2];  {将当前序列在中间位置的数定义为中间数}
    repeat
      while a[i]<mid do inc(i);        {在左半部分寻找比中间数大的数}
      while a[j]>mid do dec(j);        {在右半部分寻找比中间数小的数}
      if i<=j then begin         {若找到一组与排序目标不一致的数对则交换它们}
                     swap(a[i],a[j]);
                     inc(i);dec(j);  {继续找}
                   end;
    until i>j;
    if l<j then qsort(l,j);         {若未到两个数的边界，则递归搜索左右区间}
    if i<r then qsort(i,r);
  end;{sort}
```

从输出的数据可以发现,对较坏的初始数据如第一组输入数据,快速排序在交换数据的次数方面已给出了相当好的结果.时间的复杂性是 $O(n\log^2 n)$,速度快,但它也是不稳定的排序方法.

## 6、堆排序

(一)．堆排序思想

堆排序是一种树形选择排序，在排序过程中，将 A[1..n]看成是完全二叉树的顺序存储结构，利用完全二叉树中双亲结点和孩子结点之间的内在关系来选择最小的元素。

(二)．堆的定义：n 个元素的序列 $K_1$，$K_2$，$K_3$，…$K_n$ 称为堆，当且仅当该序列满足特性：$K_i \leqslant K_{2i}$ ， $Ki \leqslant K_{2i+1}$（$1 \leqslant i \leqslant n/2$）

堆实质上是满足如下性质的完全二叉树：树中任一非叶子结点的关键字均大于等于其孩子结点的关键字。例如序列{1,35,14,60,61,45,15,81}就是一个堆，它对应的完全二叉树如下图 1 所示。这种堆中根结点（称为堆顶）的关键字最小，我们把它称为小根堆。反之，若完全二叉树中任一非叶子结点的关键字均大于等于其孩子的关键字，则称之为大根堆。

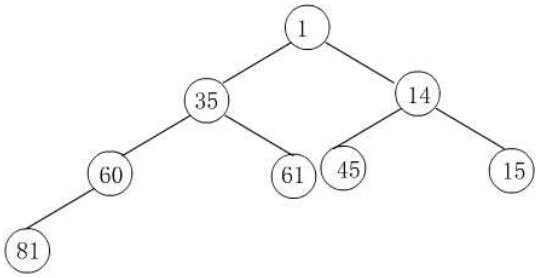



图 4_1 最小堆

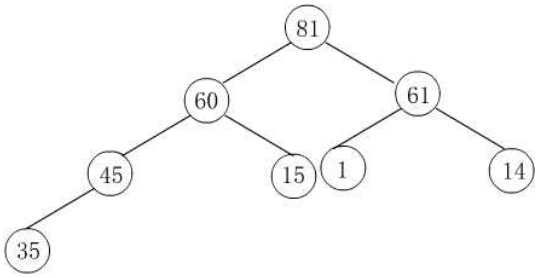



图 4_2 最大堆

存储结构：

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 35 | 14 | 60 | 61 | 45 | 15 | 81 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 81 | 60 | 61 | 45 | 15 | 1 | 14 | 35 |

(三)．堆排序过程（以最大堆为例）

（1）调整堆

假定待排序数组 A 为{20，12，35，15，10，80，30，17，2，1}（n=10），初始完全树状态为：

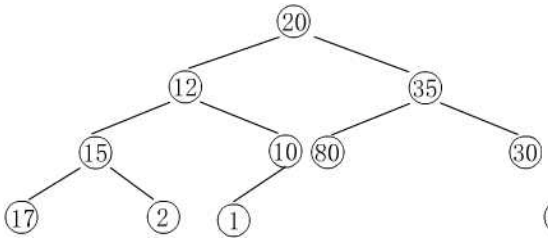



图 4_3

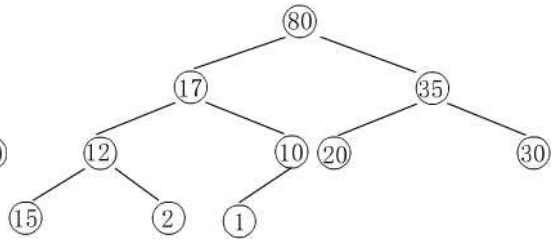



图 4_4

结点（20）、(35)、(12)等，值小于其孩子结点，因此，该树不属最大堆。

为了将该树转化为最大堆，从后往前查找，自第一个具有孩子的结点开始，根据完全二叉树性质，这个元素在数组中的位置为 i=[n/2]，如果以这个结点为根的子树已是最大堆，则此时不需调整，否则必须调整子树使之成为堆。随后，继续检查以 i-1、i-2 等结点为根的子树，直到检查到整个二叉树的根结点（i=1），并将其调整为堆为止。

调整方法：由于 A[i]的左、右子树均已是堆，因此 A[2i]和 A[2i+1]分别是各自子树中关键字最大的结点。若 A[i]不小于 A[2i]和 A[2i+1]，则 A[i]没有违反堆性质，那么以 A[i]为根的子树已是堆，无须调整；否则必须将 A[i]和 A[2i]与 A[2i+1]中较大者（不妨设为 A[j]）进行交换。交换后又可能使结点 A[j]违反堆性质，同样由于该结点的两棵子树仍然是堆，故可重复上述的调整过程，直到当前被调整的结点已满足堆性质，或者该结点已是叶子结点为止。

以上图为例，经过调整后，最大堆为：{80，17，35，12，10，20，30，15，2，1}。如图 4_4 所示。此堆作为排序的初始无序区。

（2）选择、交换、调整

①将建成的最大堆作为初始无序区。

②将堆顶元素（根）A[1]和 A[n]交换，由此得到新的无序区 A[1..n-1]和有序区 A[n]，且满足 A[1..n-1]≤A[n]

③将 A[1..n-1]调整为堆。

④再次将 A[1]和无序区最后一个数据 A[n-1]交换，由此得到新的无序区 A[1..n-2]和有序区 A[n-1..n]，且仍满足关系 A[1..n-2]≤A[n-1..n]，同样要将 A[1..n-2]调整为堆。直到无序区只有一个元素 A[1]为止。

说明：如果需要生成降序序列，则利用最小堆进行操作。

```
procedure editheap(i:integer; s:integer);      {堆的调整}
  var j:integer;
  begin
       if 2*i<=s then                    {如果该结点为叶子，则不再继续调整}
          begin
                a[0]:=a[i]; j:=i;
                if a[2*i]>a[0] then begin a[0]:=a[2*i]; j:=2*i; end;
                if (2*i+1<=s) and (a[2*i+1]>a[0]) then
                   begin a[0]:=a[2*i+1]; j:=2*i+1; end; {获取最大值}
                if j<>i then
                   begin
                        a[j]:=a[i]; a[i]:=a[0]; {更新子树结点}
                        editheap(j,s);          {调整左右孩子}
                   end;
          end;
  end;
procedure buildheap;                            {堆的建立}
  var i,j:integer;
  begin
       for i:=n div 2 downto 1 do                {从后往前，依次调整}
           begin
                a[0]:=a[i]; j:=i;
                if a[2*i]>a[0] then begin a[0]:=a[2*i]; j:=2*i; end;
                if (2*i+1<=n) and (a[2*i+1]>a[0]) then
                   begin a[0]:=a[2*i+1]; j:=2*i+1; end;
                if j<>i then
                   begin
                        a[j]:=a[i]; a[i]:=a[0];
                        editheap(j,n);
                   end;
           end;
  end;
```

```
procedure heapsort(n:integer);          {堆排序}
  var k,i,j:integer;
  begin
    buildheap;
    for i:=n downto 2 do
       begin
            a[0]:=a[i];  a[i]:=a[1]; a[1]:=a[0];
            editheap(1,i-1);
       end;
  end;
```

**7.归并排序**

所谓归并排序是指将两个或两个以上有序的数列(或有序表),合并成一个仍然有序的数列(有序表).这样的排序方法经常用于多个有序的数据文件归并成一个有序的数据文件.归并排序的算法比较简单。

**基本思想方法是**:

(1) 假设已经有两个有序数列,分别存放在两个数组s,r中;并设I,j分别为指向数组的第一个单元的下标;s有n个元素,r有m个元素.

(2) 再另设一个数组a,k指向该数组的第一个单元下标.

(3) 算法分析

```
procedure merge(s,r,a,I,j,k);
  begin
    i1:=I;j1:=j;k1:=k;
    while(i1<n) and (j1<m)do
      if s[i1]<=r[j1] then
            begin a[k]:=s[i1];i1:=i1+1;k:=k+1; end;
      else  begin a[k]:=r[j1];j1:=j1+1;k:=k+1; end;
    while i1<n do
      begin
        a[k]:=s[i1];i1:=i1+1;k:=k+1;
      end;
    while j1<=m do
      begin
        a[k]:=r[j1];j1:=j1+1;k:=k+1;
      end;
  end;
```

归并排序算法比较容易理解,完整的程序请读者自己完成.这种排序算法,经常用于解决实际生活中的一些问题,例如:数学中的两个多项式合并同类项的问题.

**【例1】有两个多项式:y1=3x5-3x5+7x2-5x-2和y2=7x8+4x6-2x5+3x4-6x3+2x-9编一个程序,求两个多项式的和.**

**问题分析:**

(1) 多项式加法问题,就是多项式同类项合并问题,也就是两个有序表的归并排序问题.

(2) 问题关键是如何表示两个多项式,这里我们可以借助记录型数组:数组的每个单元表示一个项,它有两个域系数和指数.

(3) 利用排序的归并算法,进行多项式加法运算,注意当其中某一项的和为 0 时的处理方法.

**【参考程序】**

```
program p12-16;
const max=100;
type number=record
              coef:integer;
              exp:0..100;
            end;
      arr=array[1..max] of number;
var s,r,a:arr;
    n,m,j,k,x,y:integer;
  procedure input(var h:arr;t:integer);
  var k,j:integer;
    begin
      for k:=1 to t do
        read(h[k].coef,' x' ,h[k].exp,' ' );
        writeln:
      end;
BEGIN
  write( 'input number n,m:' );
  reak(n,m);
  writeln;
  input(s,n);
  input(r,m);
  x:=1;y:=1;k:=1;
  while(x<=n)and (y<=m) do
    begin
      if s[x].exp=r[y].exp then
        begin
          j:=s[x].coef+r[y].coef;
          if j<>0 then  begin
                          a[k].exp:=s[x].exp;
                          a[k].coef:=j;
                          x:=x+1;y:=y+1;k:=k+1;
                        end;
          else begin x:=x+1;y:=y+1;end;
        end;
      else if s[x].exp>r[y].exp then
              begin
                a[k].exp:=s[x].exp;
                a[k].coef:=s[x].coef;
                x:=x+1;k:=k+1;
              end
```

```
            else begin
                    a[k].exp:=r[y].exp;
                    a[k].coef:=r[y].coef;
                    y:=y+1;k:=k+1;
                  end;
    end;
  while(x<=n)and(y>m) do
    begin
      a[k].exp:=r[y].exp;
      a[k].coef:=r[y].coef;
      x:=x+1;k:=k+1;
    end;
  while(x<=m)and(x>n)do
    begin
      a[k].exp:=r[y].exp;
      a[k].coef:=r[y].coef;
      y:=y+1;k:=k+1;
    end;
  write( 'y3=' );
  for j:=1 to k-1 do
    srite(a[i],coef,' x' ,a[j].exp,' ' );
  writeln;
END.
```

程序运行结果:

```
input number n,m: 5 7
3 5
-3 4
7 2
-5 1
-2 0
y=3X5-3X4 7X2-5X1 -2X0
7 8
4 6
-2 5
3 4
-6 3
2 1
-9 0
y=7X8 4X6 -2X5 3X4 -6X3 2X1 -9X0
y3=7X8 4X6 1X5 -6X3 7X2 -3X1 -11X0
```

以上我们介绍了数据的排序处理方法,有关数据插入,删除的操作在前面的程序设计语言中已经涉及到,其操作也比较简单.

# 第六章　排列和组合

在我们解决实际问题中,有许多问题需要将结果一一列举出来.如,四个人照相,问他们有多少种排列方法,并输出排列的形式,又如 10 个朋友见面,每两个人握一次手,共有多少种方法,输出所有握手情况等,诸如此类问题很多,可以用数学中的排列和组合问题解决。

在学习排列组合以前，我们先学习加法原理和乘法原理。

## 一、加法原理与乘法原理

### （一）加法原理

**引例 1** 从甲地去乙地，可以乘火车，可以乘汽车，还可以乘轮船。一天中，火车有 4 班，汽车有 2 班，轮船有 2 班，轮船有 3 班，那一天中乘坐这些交通工具从甲到乙地有多少种不同的选择

在一天中，从甲地到乙地乘火车有 4 种选择。乘汽车有 2 种选择，乘轮船 有 3 种选择，以上无论选择了哪一种方法，都有可以从甲地到达乙地。因此，一天当中乘坐这些交通工具从甲地到乙地的不同选择共有：

$$4+2+3=9\text{（种）}$$

把“从甲地到乙地”看成为“完成一件事”，完成它有三类方法（火车、汽车、轮船），每类方法都可以独立完成。

第一类有 4 种方法（火车有 4 班）

第二类有 2 种方法（汽车有 2 班）

第三类有 3 种方法（轮船有 3 班）

因此完成一件事（从甲地到乙地）共有：4+2+3=9 种不同的方法。

**加法原理**

**加法原理** 做一件事，完成它有 n 类方法，第一类有 m1 种，第二类有 m2 种，……，第 n 类有 mn 种，那么样完成这件事共有：

$$N=m_1+m_2+\cdots+m_n$$

加法原理的特点是：分类独立完成。

**例 1** 书架上层有不同的数学书 15 本，中层有不同的语文书 18 本，下层有不同的物理 7 本。现从其中任取一本书，问有多少种不同的取法？

解：从书架上任取一本书，有三类取法：第一类取法是从书架上层取出一本数学系书，可以从 15 本中任取一种，有 15 种取法；第二类取法是从书架的第二层取出一本语文书，可以从 18 本中任取一种，有 18 种取法；第三类取法是从书架的下层取出一本物理书，可从 7 本中任取一种，有 7 种取法。只在书架上任意取出一本书，任务即完成，根据加法原理，不同的取法一共有：

$$N=m_1+m_2+m_3=15+18+7=40\text{（种）}$$

**例 2** 某班同学分成甲、乙、丙、丁 4 个小组，甲组 9 人，乙组 11 人，丙组 10 人，丁组 9 人。现要求该班选派一人去参加某项活动，问有多少种不同的选法？

解：该班同学分成甲、乙、丙、丁四个小组，从任何一个小组中选出一名同学去参加活动，则任务完成。在甲组有 9 种选法，乙组有 11 种选取法，丙组有 10 种选法，丁组有 9 种选法，所以一共有：

$$N=9+11+10+9=39$$

**巩固性练习：**

1、一件工作可以用 2 种方法完成，有 5 人会用第一种方法，另外有 4 人会用第二种方法完成，要选出 1 个人来完成这件工作，共有多少种选法？

2、一个学生要从 2 本科技书，2 本政治书，3 本文艺本中任取一本，共有多少种不同的取法？

## （二）乘法原理：

由 A 地去 C 地，中间必须经过 B 地，且已知由 A 地到 B 地有 3 条路可走，再由 B 地到以地有 2 条路可走，那么由 A 地经 B 地到 C 地有多少种不同的走法？

这里，从 A 地到 C 地不能由一个步骤直接到达，必须经过 B 地这一步骤，从 A 地到 B 地有 3 种不同的走法，分别用 $a_1$、$a_2$、$a_3$ 表示，而从 B 地到 C 地有 2 种不同的走法，分别用 $b_1$、$b_2$ 表示。所以从 A 地经 B 地到 C 地的全部走法有：

$$a_1b_1；a_1b_2；a_2b_1；a_2b_2；a_3b_1；a_3b_2，$$

共计 6 种。就是从 A 地到 C 地的 3 种走法与从 C 地到 B 地的 2 种走法的乘积，即：3×2=6（种）。

把“从 A 地到 C 地”看成完成一件事，那么完成这件必须分二个步骤，二个步骤都完成了，才能完成这个事。

第一个步骤有 3 种方法（从 A 地到 B 地）

第二个步骤有 2 种方法（从 B 地到 C 地）

因此“完成一件事”（从 A 地到 C 地）共有 3×2=6（种）不同的方法

**乘法原理**

**乘法原理** 做一件事，完成它需要 n 个先后步骤，做第一步有 $m_1$ 种不同的方法，做第二步有 $m_2$ 种不同的方法，……，做第 n 步有 $m_n$ 种不同的方法，那么完成这件事共有

$$N=m_1\times m_2\times\cdots m_n$$

种不同的方法。

乘法原理的特点是：分步依次完成

**例 3** 书架上层有不同的数学书 15 本，中层有不同的语文书 18 本，下层有不同的物理 7 本，从中取出数学、语文、物理书各一本，问有多少种不同的取法？

解：从书上取数学、语文、物理书各 1 本，可以分成 3 个步骤完成：第一步取数学书 1 本，有 15 咱不同的取汪；第二不取语文书 1 本，有 18 种不同的取法；第三步取物理书 1 本，有 7 种不同的取本，符合乘法原理的条件。利用乘法原理，得到：N=15×18×7=1 890（种）不同的取法。

**例 4** 某农场要在 4 种不同类型的土地上，引种试验 A、B、C、D 等 4 种不同品种的小麦，问有多少种不同的试验方案？

解：第一步　先考虑 A 种小麦，可在 4 种不同类型的土地中任选一种，有 4 种选法；

第二步　考虑 B 种小麦，可在剩下的 3 种不同类型的土地中任选一种，有 3 种选法；

第三步　考虑 C 种小麦，再在剩下的 2 种不同类型的土地中任选一种，有 2 种选法；

第四步　最后考虑 D 种小麦，再在剩下的 2 种不同类型的土地中任选一种，有 1 种选法。

以上四步依次完成，才算完成，依据乘法原理，可知有 4×3×2×1=24 不同的试验方案。

## 巩固性练习：

3、在一个红色口袋中，装有 20 张分别标有 1，2，…，20 的红色数字卡片，在另一个黄色口袋中，装有 10 张分别标有 1，2，…，10 的黄色数字卡片。在红色口袋中摸了一张数字卡片做被加数，在黄色口袋中摸出一张数字卡片做加数，列成加法式子，一共可以列成多少个符合要求的加法式子？

4、从 A 地到 B 地有 2 条路可通，从 B 地到 C 地有 3 条路可通，从 A 地到 C 地共有多少种不同的走法？

5、一个口袋内有 5 个小球，另一个口袋内有 4 个小球，所有这些小球的颜色互不相同。

（1）从两个口袋内任取一个小球，有多少种不同的取法？

（2）从两个口袋内各取一个小球，有多少种不同的取法？

6、从 2，3，5，7 这四个数中，取两个数出来做假分数，这些假分数有多少个？

## 二、排列问题

**引例** 有 4 人照相,按照们排列位置的不同照相,问有多少种排列方法?

这是一种乘法原理应用题,即当我们选中排列在第一位上的数后,则第 2 位有 3 种可能,而当第二位确定后,其第三位有两种可能,当第三位确定后,第四位只有一种可能,所以其排列数是:(3*2*1)*4=4!=24 种排列。

所谓排列问题一般形式是:任给 n 个不同元素，从中取出 r 个不同元素,排成一列,试问有多少种不同的排列数目,不仅要给出排列的数目且要能列举出所有不同的排列序列。当 r=n 时，称为全排列。

一般地，n 个元素全排列数是：n!=n*(n-1)*(n-2)*…*2*1

n 个元素中取出 r 个元素的排列数是：$P_n^r = n*(n-1)*\cdots(n-r+1)$

输出每种排列：输出每种排列,可以用枚举方法输出,但在 n 较大时,要枚举出 n 个元素的排列就比较困难.如 n=10,要保存每个全排列需要 10!的空间,这是内存无法完成的。

**例 5：**有 3 个人排成一个队列，问有多少种排对的方法，输出每一种方案?

分析：如果我们将 3 个人进行编号，分别为 1、2、3，显然我们列出所有的排列，123，132，213，231，312，321 共六种。可用循环枚举各种情况，参考程序：

```
program exam5;
var
    i,j,k:integer;
begin
    for I:=1 to 3 do
          for j:=1 to 3 do
        for k:=1 to 3 do
        if (i+j+k=6) and (i*j*k=6) then writeln(i,j,k);
end.
```

上述情况非常简单，因为只有 3 个人，但当有 N 个人时怎么办？显然用循环不能解决问题。下面我们介绍一种求全排列的方法。

设当前排列为 $P_1\ P_2\ ,\cdots,P_n$，则下一个排列可按如下算法完成：

1．求满足关系式 $P_{j-1} < P_j$ 的 J 的最大值，设为 I，即

I=max{j ｜ $P_{j-1} < P_j$， j = 2..n}

2．求满足关系式 $P_{i-1} < P_k$ 的 k 的最大值，设为 j，即

J=max{K ｜ $P_{i-1} < P_k$， k = 1..n}

3．$P_{i-1}$ 与 $P_j$ 互换得 (P) = $P_1\ P_2\ ,\cdots,P_n$

4．(P) = $P_1\ P_2\ ,\cdots,\ P_{i-1}\ P_i,\cdots,\ P_n$ 部分的顺序逆转，得 $P_1\ P_2\ ,\cdots,\ P_{i-1}\ P_n\ P_{n-1},\cdots,\ P_i$ 便是下一个排列。

例：设 $P_1\ P_2\ P_3\ P_4$ =3421

1．I= max{j ｜ $P_{j-1} < P_j$， j = 2..n} = 2

2．J=max{K ｜ $P_{i-1} < P_k$， k =1..n} = 2

3．$P_1$ 与 $P_2$ 交换得到 4321

4．4321 的 321 部分逆转得到 4123 即是 3421 的下一个排列。

程序设计如下：

```
program exam5;
const
     maxn      = 100;
var   i,j,m,t : integer;
```

```
      p       : array[1..maxn] of integer;
      count   :integer;          {排列数目统计变量}
begin
    write('m:');readln(m);
    for i:=1 to m do begin p[i]:=i; write(i) end;
    writeln;
    count:=1;
    repeat
          {求满足关系式 Pj-1 < Pj 的 J 的最大值，设为 I}
      i:=m;
      while (i>1) and (p[i-1]>=p[i]) do dec(i);
      if i=1 then break;
         {求满足关系式 Pi-1 < Pk 的 k 的最大值，设为 j}
      j:=m;
      while (j>0) and (p[i-1]>=p[j]) do dec(j);
      if j=0 then break;
          {Pi-1 与 Pj 互换得 (P) = P1 P2 ,…,Pm}
      t:=p[i-1];p[i-1]:=p[j];p[j]:=t;
          {Pi,…, Pm 的顺序逆转}
      for j:=1 to (m-i+1) div 2 do begin
          t:=p[i+j-1];p[i+j-1]:=p[m-j+1];p[m-j+1]:=t
      end;
    {打印当前解}
    for i:=1 to m do write(p[i]);
    inc(count);
    writeln;
    until false;
    writeln(count)
End.
```

## 三、组合问题

组合一般是指从 n 个不同元素中取 r 个元素的不同组合的数目。

它与排列算法不同在于前者不仅需要考虑不同元素,而且需要考虑元素所在的不同位置；组合则仅仅考虑不同元素组合,不考虑元素位置。10 个人握手 A 与 B 握手与 B 与 A 握手认为是同一种方式,计数 1 次;而两个人照相则是:A 在左 B 在右与 B 在左 A 在右是两张不同的照片,计数 2 次。

组合数的计算公式: C(n,r) =n(n-1)(n-2)……(n-r+1)！/r!

组合数和排列数之间的关系：$p_n^r$=C（n,r）*r!

可以理解为：从 n 个元素中取 r 中元素的排列分为两步：（1）从 n 个元素中取 r 中元素的组合，（2）每一种组合，求对 r 个元素全排列。由乘法原理即可得上述公式。

例如当 n=4,r=3,用 C(n,r)表示为 C(4,3),即从 4 个元素中取 3 个元素的组合数,结果为 4 种组合.设 4 个元素用 1,2,3,4 表示,4 种组合是:{1 2 3} {1 2 4} {1 3 4} {2 3 4 }，每一种组合有 3!个排列，共有 24 种排列。

对于组合数的计算,完全可以利用组合公式进行计算,那么如何逐个生成元素的每种组合呢?问题关键在于探讨生成组合中元素排列顺序的规律。

**例 6**：求 N 个人选取 M 个人出来做游戏，共有多少种取法？例如：N=4，M=2 时，有 12，13，14，23，24，34 共六种。

分析：因为组合数跟顺序的选择无关。因此对同一个组合的不同排列，只需取其最小的一个（即按从小到大排序）。因此，可以设计如下算法：

1．最后一位数最大可达 N，倒数第二位数最大可达 N-1，…，依此类推，倒数第 K 位数最大可达 N-K+1。

若 R 个元素组合用 $C_1C_2 \cdots C_R$ 表示，且假定 $C_1<C_2< \cdots <C_R$，$C_R<=N-R+I$，　I=1,2,…,R。

2．当存在 $C_j<N-R+J$ 时，其中下标的最大者设为 I，即

　　$I=\max\{J \mid C_j<N-R+J\}$，则作 $C_i := C_i +1$,与之对应的操作有

　　$C_{i+1} := C_i +1$ ，$C_{i+2} := C_{i+1}+1$ ，…. ，$C_R := C_{R-1} +1$

参考程序：

```
program exam6;
const maxn=10;
var  i,j,n,m :integer;
    c   :array[1..maxn]of integer;      {c 数组记录当前组合}
Begin
        Write('n & m:'); readln(n,m);
    for i:=1 to m do begin              {初始化，建立第一个组合}
      c[i]:=i;
      write(c[i]);
    end;
    writeln;
    while c[1]<n-m+1 do begin
      j:=m;
      while (c[j]>n-m+1) and ( j>0) do dec(j);      {求 I=max{J | Cj<N-R+J} }
      c[j]:=c[j]+1;
      for i:=j+1 to m do c[i]:=c[i-1]+1;                {建立下一个组合}
      for i:=1 to m do write(c[i]);writeln              {输出}
    end;
End.
```

## 【上机练习】

**1**、如果直角三角形的边长均小于 10，且可以表示为：

$m^2+n^2$，$m^2-n^2$，2mn(m>n>≥1,m，n∈N*)，问有多少个这样的直角三角形？

**[算法分析]**由已知得 $m^2+n^2<100$，所以 m≤9。

**2**、用数字 0，1，2，3 能组成多少个三位数？

# 第七章　递推算法

递推法是一种重要的数学方法，在数学的各个领域中都有广泛的运用，也是计算机用于数值计算的一个重要算法。这种算法特点是：一个问题的求解需一系列的计算，在已知条件和所求问题之间总存在着某种相互联系的关系，在计算时，如果可以找到前后过程之间的数量关系（即递推式），那么，从问题出发逐步推到已知条件，此种方法叫逆推。无论顺推还是逆推，其关键是要找到递推式。这种处理问题的方法能使复杂运算化为若干步重复的简单运算，充分发挥出计算机擅长于重复处理的特点。

递推算法的首要问题是得到相邻的数据项间的关系（即递推关系）。递推算法避开了求通项公式的麻烦，把一个复杂的问题的求解，分解成了连续的若干步简单运算。一般说来，可以将递推算法看成是一种特殊的迭代算法。

**【例 1】数字三角形。**如下所示为一个数字三角形。请编一个程序计算从顶到底的某处的一条路径，使该路径所经过的数字总和最大。只要求输出总和。

```
      7
    3   8
  8   1   0
2   7   4   4
4   5   2   6   5
```

1、 一步可沿左斜线向下或右斜线向下走；

2、 三角形行数小于等于 100；

3、 三角形中的数字为 0，1，…，99；

测试数据通过键盘逐行输入，如上例数据应以如下所示格式输入：

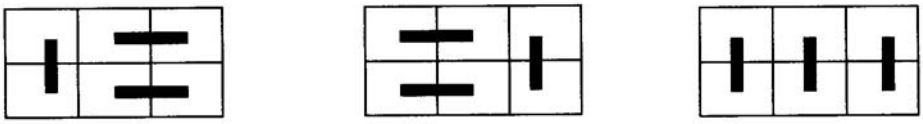
```
5
7
3 8
8 1 0
2 7 4 4
4 5 2 6 5
```

**【算法分析】**

此题解法有多种，从递推的思想出发，设想，当从顶层沿某条路径走到第 I 层向第 I+1 层前进时，我们的选择一定是沿其下两条可行路径中最大数字的方向前进，为此，我们可以采用倒推的手法，设 a[i，j]存放从 i，j 出发到达 n 层的最大值，则 a[i，j]=max{a[i，j]+a[i+1，j]，a[i，j]+a[i+1，j+1]}，a[1，1] 即为所求的数字总和的最大值。

```
program ex11_9;
  var n,j,i:integer;
      a:array[1..100,1..100] of integer;
begin
  read(n);
  for i:=1 to n do
    for j:=1 to i do
        read(a[i,j]);
    for i:=n-1 downto 1 do
      for j:=1 to i do
        begin
          if a[i+1,j]>=a[i+1,j+1] then a[i,j]:= a[i,j]+a[i+1,j]
                                  else a[i,j]:=a[i,j]+a[i+1,j+1];
        end;
  writeln(a[1,1]);
end.
```

**【例 2】 有 2×n 的一个长方形方格，用一个 1*2 的骨牌铺满方格。例如 n=3 时，为 2*3 方格。此时用一个 1*2 的骨牌铺满方格，共有 3 种铺法：**



编写一个程序，试对给出的任意一个 n(n>0)，输出铺法总数。

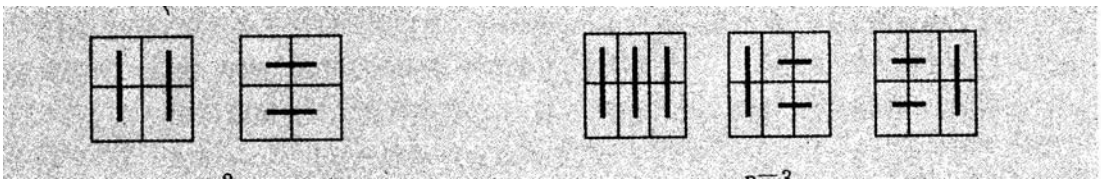
**【算法分析】**

（1）面对上述问题，如果思考方法不恰当，要想获得问题的解答是相当困难的。可以用递推方法归纳出问题解的一般规律。

（2）当 n=1 时，只能是一种铺法

如左图，铺法总数表示为 X1=1；

（3）当 N=2 时：骨牌可以两个并列竖排，也可以并列横排，再无其他方法，如下左图所示，因此，铺法总数表示为 X2=2；



（4）当 N=3 时：骨牌可以全部竖排，也可以认为在方格中已经有一个竖排骨牌，则需要在方格中排列两个横排骨牌（无重复方法），若已经在方格中排列两个横排骨牌，则必须在方格中排列一个竖排骨牌。如上右图，再无其他排列方法，因此铺法总数表示为 x3=3.

由此可以看出，当 n=3 时的排列骨牌的方法数是 n=1 和 n=2 排列方法数的和。

(5) 推出一般规律：对一般的 n，要求 $X_N$ 可以这样来考虑，若第一个骨牌是竖排列放置，剩下有 n-1 个骨牌需要排列，这时排列方法数为 $X_{N-1}$；若第一个骨牌是横排列，整个方格至少有 2 个骨牌是横排列（1*2 骨牌），因此剩下 N-2 个骨牌需要排列，这是骨牌排列方法娄为 $X_{N-2}$。从第一骨牌排列方法考虑，只有这两种可能，所以有：

$$X_n=X_{n-1}+X_{n-2} \quad (N>2)$$
$$x1=1 \qquad\qquad (A)$$
$$x2=2$$

(A)就是问题求解的递推公式。任给 N 都可以从中获得解答。例如 N=5，

X3=X2+X1=3

X4=X3+X2=5

X5=X4+X3=8

下面是输入 N，输出 X1~XN 的 Pascal 程序：

```
program p12_20;
  var  x,y,z:longint;
       i,n:integer;
  begin
      write('Input n:');
      read(n);
```

```
        x:=0;
        y:=1;
        for i:=1 to n do
           begin
              z:=y+x;
              writeln('x[',i:2,']=',z);
              x:=y;y:=z;
           end;
     end.
```

下面是运行程序输入 n=30，输出的结果：

```
  input n:30
     x[1]=1
     x[2]=2
     x[3]=3
     x[4]=5
     x[5]=8
.............................
     x[28]=514229
     x[29]=832040
     x[30]=1346269
```

问题的结果就是有名的斐波那契数。

## 【例 3】棋盘格数

设有一个 N*M 方格的棋盘（ 1≤ N≤100，1≤M≤100）。求出该棋盘中包含有多少个正方形、多少个长方形（不包括正方形）。

例如：当 N=2， M＝3 时：

正方形的个数有 8 个：即边长为 1 的正方形有 6 个；边长为 2 的正方形有 2 个。

长方形的个数有 10 个：即 2*1 的长方形有 4 个：1*2 的长方形有 3 个：3*1 的长方形有 2 个：3*2 的长方形有 1 个：

程序要求：输入：N，M

输出：正方形的个数与长方形的个数

如上例：输入：2　3

输出：8，10

**【算法分析】**

**1.计算正方形的个数 $s_1$**

边长为 1 的正方形个数为 n*m

边长为 2 的正方形个数为(n-1)*(m-1)

边长为 3 的正方形个数为(n-2)*(m-2)

……

边长为 min{n，m}的正方形个数为(m-min{n，m}+1)*(n-min{n，m}+1)

根据加法原理得出

$$s_1=\sum_{i=0}^{\min\{m,n\}-1}(n-i)*(m-i)$$

**2.长方形和正方形的个数之和 s**

宽为 1 的长方形和正方形有 m 个，宽为 2 的长方形和正方形有 m-1 个，-------，宽为 m 的长方形和正方形有 1 个；

长为 1 的长方形和正方形有 n 个，长为 2 的长方形和正方形有 n-1 个，-------，长为 n 的长方形和正方形有 1 个；

根据乘法原理

$$s=(1+2+\cdots\cdots+n)*(1+2+\cdots\cdots+m)=\frac{(1+n)*(1+m)*n*m}{4}$$

**3.长宽不等的长方形个数 $s_2$**

显然，$s_2=s-s_1$

$$=\frac{(1+n)*(1+m)*n*m}{4}-\sum_{i=0}^{\min\{m,n\}-1}(n-i)*(m-i)$$

由此得出算法：

```
program Qex9;
var m,n,m1,n1,s1,s2:longint;
begin
  readln(m,n);                                    {计算正方形的个数 s1}
  m1:=m;n1:=n;
  s1:=m1*n1;
  while (m1<>0)and(n1<>0)do
    begin
      m1:=m1-1;n1:=n1-1;
      s1:=s1+m1*n1;
    end;
  s2:=((m+1)*(n+1)*m*n) div 4-s1;                 { 计算长方形的个数 s2}
  writeln(s1,' ',s2);
end.
```

**【例 4】贮油点**

一辆重型卡车欲穿过 1000 公里的沙漠，卡车耗汽油为 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若干个贮油点，使卡车能顺利穿过沙漠。试问司机如怎样建立这些贮油点？每一贮油点应存储多少汽油，才能使卡车以消耗最少汽油的代价通过沙漠？

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。格式如下：

```
No.                    Distance(k.m.)          Oil(litre)
```

| | | |
|---|---|---|
| 1 | × × | × × |
| 2 | × × | × × |
| … | … … | … … |

**【算法分析】**

设 Way[I]——第 I 个贮油点到终点（I=0）的距离；

oil[I]——第 I 个贮油点的贮油量；

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 19 表示倒推时的返回点。

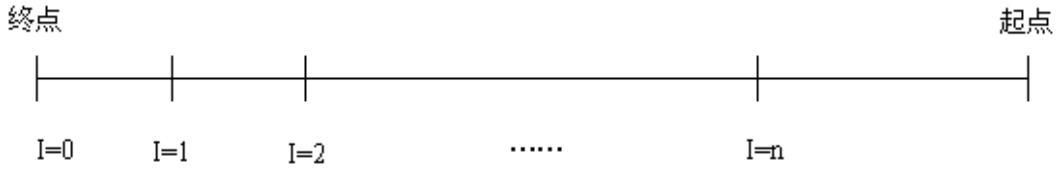



图 19 倒推过程

从贮油点 I 向贮油点 I+1 倒推的方法是：卡车在贮油点 I 和贮油点 I+1 间往返若干次。卡车每次返回 I+1 点时应该正好耗尽 500 公升汽油，而每次从 I+1 点出发时又必须装足 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≦I≦n-1）。具体来说，第一个贮油点 I=1 应距终点 I=0 处 500km，且在该点贮藏 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

Way[I]=500；oil[I]=500；

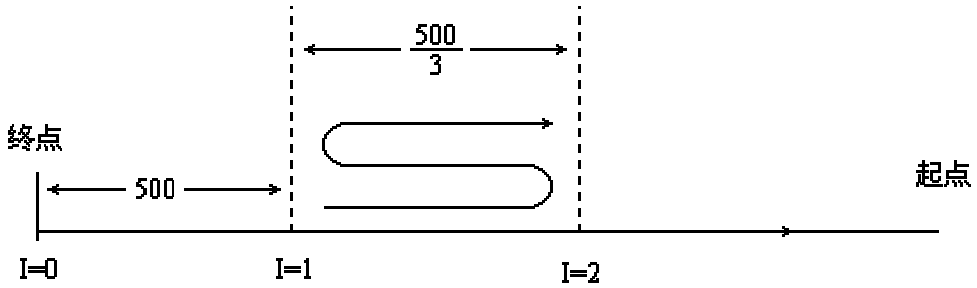



图 20 倒推到第二步

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 $d_{12}$=500/3km，Way[2]=Way[1]+$d_{12}$=Way[I]+500/3

此时的状况如图 20 所示。

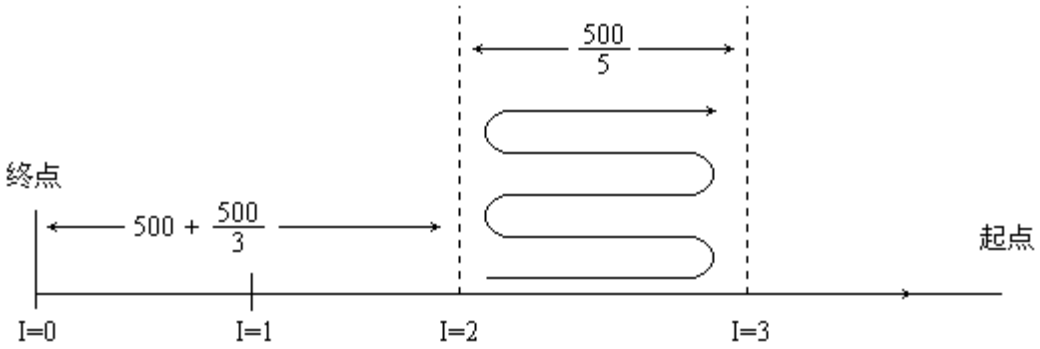



图 21 倒推到第三步

为了在 I=2 处贮藏 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。路途耗油亦应 500 公升，即 $d_{23}$=500/5，

Way[3]=Way[2]+$d_{23}$=Way[2]+500/5；

此时的状况如图 21 所示。

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)，

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

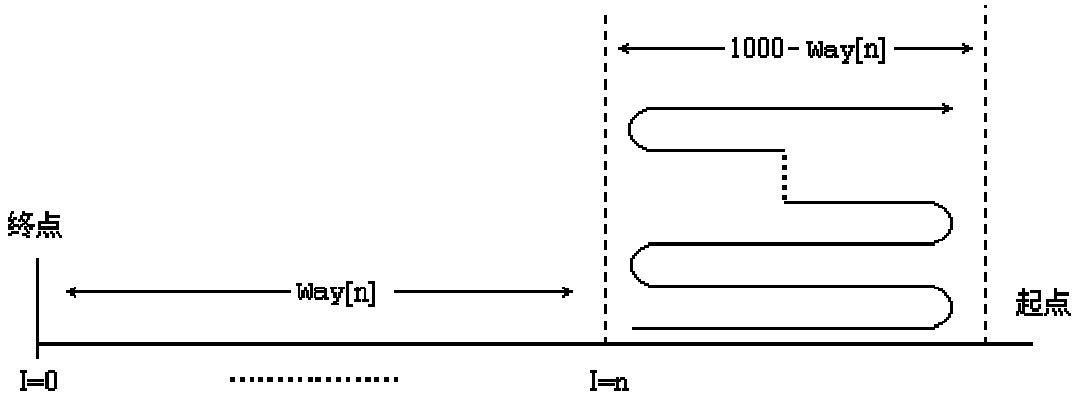



图 22 倒推到第 n 步

此时的状况如图 22 所示。

最后，I=n 至始点的距离为 1000-Way[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-Way[n]）*(2n+1)，即始点藏油为 oil[n]+(1000-Way[n])*(2n+1)。

**【参考程序】**

```
program Oil_lib;
```

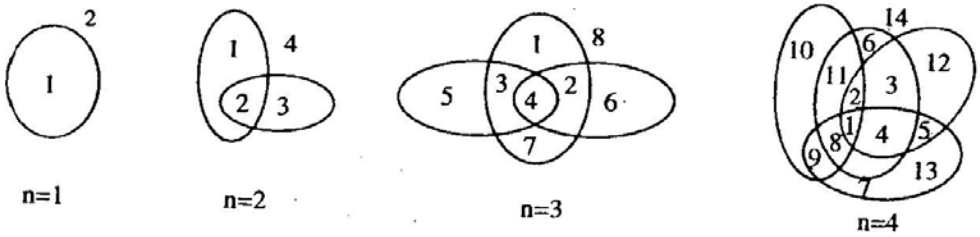
```
var
  K: Integer;     {贮油点位置序号}
  D,                  {累计终点至当前贮油点的距离}
  D1: Real;         {I=n 至终点的距离}
  Oil, Way: array [1 .. 10] of Real;
  i: Integer;
begin
  Writeln(‘No.’, ‘Distance’:30, ‘Oil’:80);
  K := 1;
  D := 500;       {从 I=1 处开始向终点倒推}
  Way[1] := 500;
  Oil[1] := 500;
  repeat
    K := K + 1;
    D := D + 500 / (2 * K - 1);
    Way[K] := D;
    Oil[K] := Oil[K - 1] + 500;
  until D >= 1000;
  Way[K] := 1000;                  {置始点到终点的距离值}
  D1 := 1000 - Way[K - 1];          {求 I=n 处至至点的距离}
  Oil[K] := D1 * (2 * k + 1) + Oil[K - 1];  {求始点贮油量}
  {由始点开始，逐一打印至当前贮油点的距离和贮油量}
  for i := 0 to K do
    Writeln(i, 1000 - Way[K - i]:30, Oil[K - i]:80);
end.
```

## 【上机练习】

1、移梵塔。有三根柱 A，B，C 在柱 A 上有 N 块盘片，所有盘片都是大的在下面，小片能放在大片上面。现要将 A 上的 N 块片移到 C 柱上，每次只能移动一片，而且在同一根柱子上必须保持上面的盘片比下面的盘片小。

问：将这 n 个盘子从 a 柱移到 c 柱上，总计需要移动多少个盘次？

2、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一点，问这些封闭曲线把平面分割成的区域个数。




3、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 3)，故 $h_5=5$。求对于一个任意的凸 n 边形相应的 $h_n$。

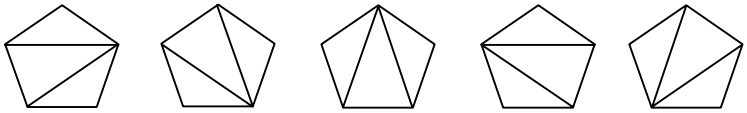


图 3

4、n 个有区别的球放到 m 个相同的盒子中，要求无一空盒，其不同的方案用 S(n,m)表示，称为第二类 Stirling 数。

5、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。试求出密蜂从蜂房 1 爬到蜂房 n 的可能路线数。（如下图）

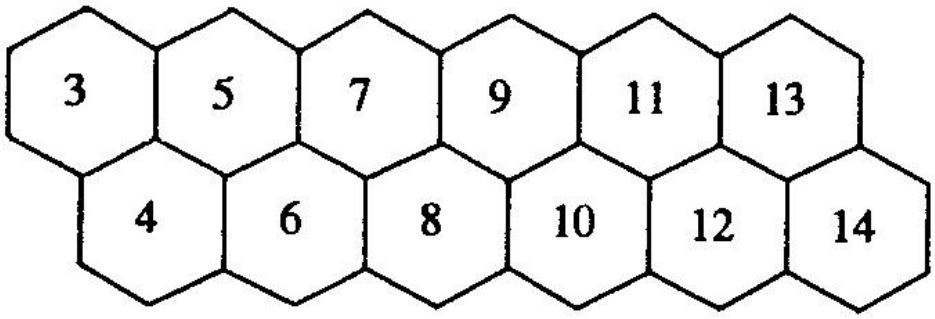



图 5－6

6、同一平面内的 n(n≤500)条直线，已知有 p(p≥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？

# 第八章　　递归算法

前面已经介绍了关于递归调用这样一种操作，而递归程序设计是 Pascal 语言程序设计中的一种重要的方法，它使许多复杂的问题变得简单，容易解决了。递归特点是：函数或过程调用它自己本身。其中直接调用自己称为直接递归，而将 A 调用 B，B 以调用 A 的递归叫做间接递归。

**【例 1】 给定 N（N>=1），用递归的方法计算 1+2+3+4+...+(n-1)+n。**

**【算法分析】**

本题可以用递归方法求解，其原因在于它符合递归的三个条件：

（1）本题是累加问题：当前和=前一次和+当前项，而前一次和的计算方法与其相同，只是数据不同————s(n)=s(n-1)+n;

(2)给定 n,所以是有限次的递归调用；

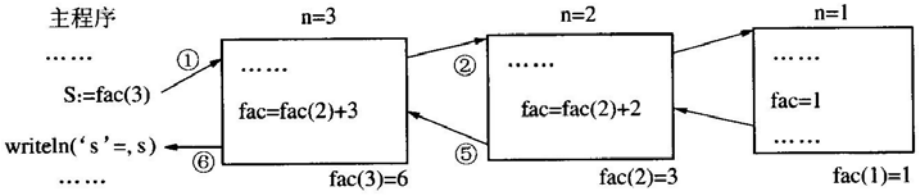
（3）结束条件是当 N=1，则 S=1。

**【参考程序】**

```
program p12_22;
    var s,t:integer;
    function fac(n:integer):integer;
      begin
         if n=1 then fac:=1 else fac:=fac(n-1)+n;
      end;
    begin
     read(t);
     s:=fac(t);
     writeln('s=',s);
    end.
```

运行程序，当 T=5 时，输出结果：S=15，其递归调用执行过程是：（设 T=3）

主程序 N=3，N=2，N=1 时，




递归调用过程，实质上是不断调用子过程或子函数的过程，由于第递归调用一次，所有子程序的变量（局部变量、变参等）、地址在计算机内部都有用特殊的管理方法——栈（先进后出）来管理，一旦递归调用结束，计算机便开始根据栈中存储的地址返回各子程序变量的值，并进行相应操作。

**【例 2】 设有 N 个数已经按从大到小的顺序排列，现在从键盘上输入 X，判断它是否在这 N 个数中，如果存在则输出：“YES” 否则输出“NO”。**

**【算法分析】**

该问题属于数据的查找问题，数据查找有多种方法，通常方法是：顺序查找和折半查找，当 N 个数排好序时，用折半查找方法速度大大加快。折半查找算法：

（1） 设有 N 个数，存放在 A 数组中，待查找数为 X，用 F 指向数据的高端，用 R 指向数据的低端，MID 指向中间：

（2） 若 X=A[MID] 输出 “YES”；

（3） 若 X《A[MID]则到数据后半段查找：R 不变，F=MID+1，计算新的 MID 值，并进行新的一段查找；

（4） 若 X》A[MID]则到数据前半段查找：F 不变，R=MID-1，计算新的 MID 值，并进行新的一段查找；

若 F》R 都没有查找到，则输出“NO”。

该算法符合递归程序设计的基本规律，可以用递归方法设计。

**【参考程序】**

```
program p12_23;
   const n=30;
   var a:array[1..n]of integer;
       f,r,x,k:integer;
  procedure search(x,top,bot:integer);
      var mid :integer;
      begin
         if top<=bot then
            begin
              mid:=(top+bot) div 2
              if x=a[mid] then writeln(x:5,mid:5,'yes')
                  else
                    if x<a[mid] then search(x,mid+1, bot)
                        else  search(x,top,mid-1);
             end
           else writeln(x:5,'no');
        end;
   begin  {主程序}
      writeln('输入 30 个从大到小顺序的数：');
      for k:=1 to n do
           read(a[k]);
      readln(x);
      f:=1;r:=n;
      search(x,f,r);
    end.
```

**【例 3】Hanoi(河内/汉诺）塔问题**

有 N 个圆盘，依半径大小（半径都不同），自下而上套在 A 柱上，每次只允许移动最上面一个盘子到另外的柱子上去(除 A 柱外，还有 B 柱和 C 柱，开始时这两个柱子上无盘子)，

但绝不允许发生柱子上出现大盘子在上，小盘子在下的情况，现要求设计将 A 柱子上 N 个盘子搬移到 C 柱去的方法。

本题是典型的递归程序设计题。

（1）当 N=1 时，只有一个盘子，只需要移动一次:A—>C;

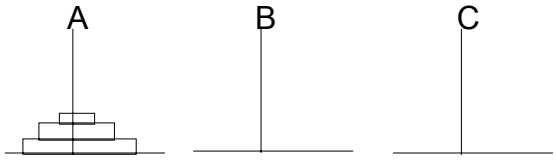
(2) 当 N=2 时，则需要移动三次：

A———— 1——> B,

A-------- 2----> C,

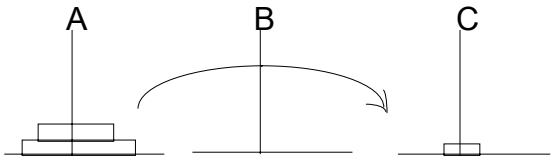
B-------- 1----> C.

(3)如果 N=3，则具体移动步骤为：

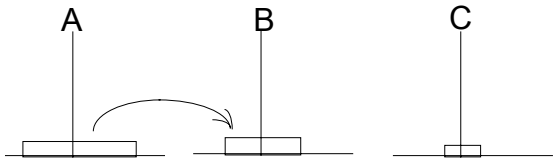



N=3 的初始状态

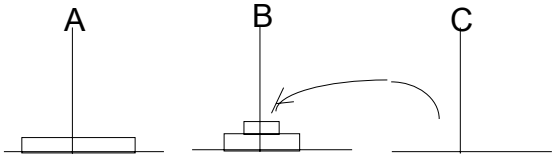
移动方法和步骤：



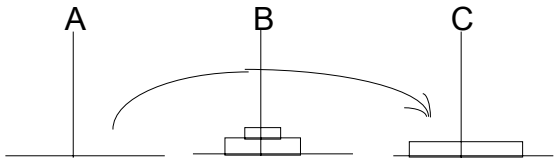

1.from A to



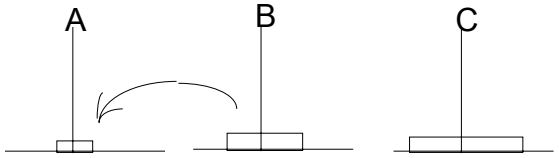

2.from A to B



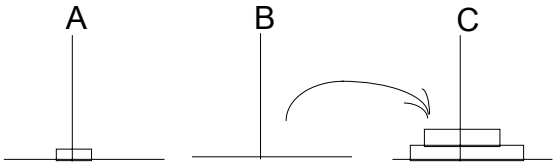

3.from C to B



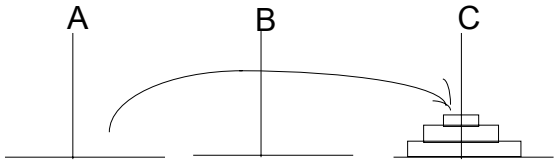

4.from A to C




5.from B to A

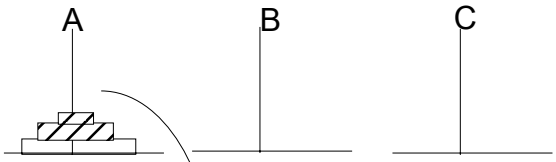



6.from B to C

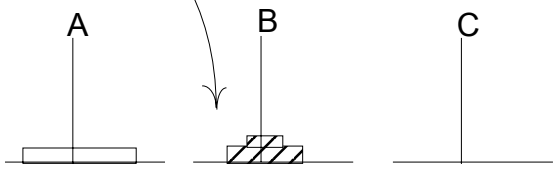



7. from A to C

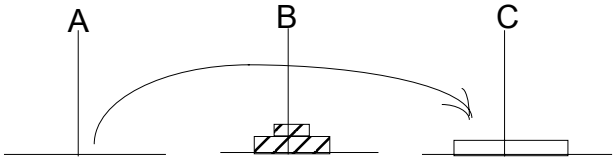
假设把第 3 步，第 4 步，第 6 步抽出来就相当于 N=2 的情况（把上面 2 片捆在一起，视为一片）：



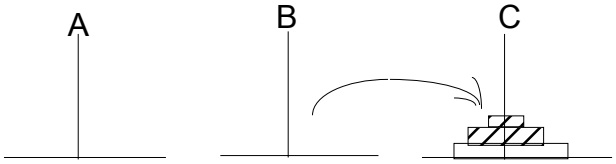

原问题：欲将A柱上的N片移到C柱上，B柱为过渡柱，记为（A，C，B）




1. 将A柱上的(N-1)片移到B上，{A为源柱，B为目标，C为过渡柱，记为（A，B，A）}




2.将A柱上剩下的1片直接移到C上；{不需调用过程}




3.将B上的(N-1)片移到C上，结束。{B为源柱，C为目标柱，A为过渡柱，记为(B,C,A)}

所以可按“N=2”的移动步骤设计：

①如果 N=0，则退出，即结束程序;否则继续往下执行;

②用 C 柱作为协助过渡，将 A 柱上的(N-1)片移到 B 柱上，调用过程 sub(n-1, a,b,c)；

③将 A 柱上剩下的一片直接移到 C 柱上;

④用 A 柱作为协助过渡，将 B 柱上的（N-1）移到 C 柱上，调用过程 sub(n-1,b,c,a)。

**【参考程序】**

```
Program Exam65;
  Var x,y,z : char;
     N, k : integer;
  Procedure  sub(n: integer; a, c , b: char);
    begin
      if n=0 then exit;
      sub(n-1, a,b,c);
      inc(k);
      writeln(k, ': from', a, '-->', c);
      sub(n-1,b,c,a);
  end;
begin
  write('n='; readln(n);
  k:=0; x:='A'; y:='B'; Z:='C';
  sub(n,x,z,y);
end.
```

程序定义了把 n 片从 A 柱移到 C 柱的过程 sub(n,a,c,b)，这个过程把移动分为以下三步来进行：

①先调用过程 sub(n-1, a, b, c)，把(n-1)片从 A 柱移到 B 柱，C 柱作为过渡柱；

②直接执行 writeln(a, '-->', c)，把 A 柱上剩下的一片直接移到 C 柱上，；

③调用 sub(n-1,b,c,a)，把 B 柱上的(n-1)片从 B 移到 C 柱上，A 柱是过渡柱。

对于 B 柱上的(n-1)片如何移到，仍然调用上述的三步。只是把(n-1)当成了 n，每调用一次，要移到目标柱上的片数 N 就减少了一片，直至减少到 n=0 时就退出，不再调用。exit 是退出指令，执行该指令能在循环或递归调用过程中一下子全部退出来。

一面的过程中出现了自己调用自己的情况，在 Pascal 中称为递归调用，这是 Pascal 语言的一个特色。对于没有递归调用功能的程序设计语言，则需要将递归过程重新设计为非递归过程的程序。

**【例 4】用递归的方法求斐波那契数列中的第 N 个数**

$$f_n=\begin{cases}0 & n=0\\ 1 & n=1\\ f_{n-1}+f_{n-2} & n>1\end{cases}$$

**【参考程序】**

```
program p12_25
   var  m,p:integer;
```

```
function fib(n:integer):integer;
   begin
      if n=0 then fib:=0
             else if n=1 then fib:=1
                         else fib:=fib(n-1)+fib(n-2);
  end;
begin
  readln(m);
  p:=fib(m);
  writeln('fib(',m,')=',p)
end.
```

输入 15　　输出 fib(15)=610

## 【上机练习】

**1、输入一串以‘！’结束的字符，按逆序输出。(用递归做)**

**2、背包问题**

**问题：**假设有 n 件质量分配为 $w_1$，$w_2$，...，$w_n$的物品和一个最多能装载总质量为 T 的背包，能否从这 n 件物品中选择若干件物品装入背包，使得被选物品的总质量恰好等于背包所能装载的最大质量，即 $w_{i1}+w_{i2}+...+w_{ik}=T$。若能，则背包问题有解，否则无解。

(例如：有 5 件可选物品，质量分别为 8 千克、4 千克、3 千克、5 千克、1 千克。假设背包的最大转载质量是 10 千克。)

**3、阿克曼（Ackmann）函数 A(x，y)中，x，y 定义域是非负整数，函数值定义为：**

$$Ack(m,n)=\begin{cases} n+1 & m=0 \\ Ack(m-1,1) & m\neq 0, n=0 \\ Ack(m-1, Ack(m, n-1)) & m\neq 0, n\neq 0 \end{cases}$$

写出计算 Ack(m，n)的递归算法程序。

4、某人写了 n 封信和 n 个信封，如果所有的信都装错了信封。求所有的信都装错信封共有多少种不同情况。

基本形式：D[1]=0;d[2]=1

递归式：d[n]= (n-1)*( d[n-1] + d[n-2])

5、有 52 张牌，使它们全部正面朝上，第一轮是从第 2 张开始，凡是 2 的倍数位置上的牌翻成正面朝下；第二轮从第 3 张牌开始，凡是 3 的倍数位置上的牌，正面朝上的翻成正面朝下,正面朝下的翻成正面朝上；第三轮从第 4 张牌开始，凡是 4 的倍数位置上的牌按上面相同规则翻转，以此类推，直到第一张要翻的牌超过 52 为止。统计最后有几张牌正面朝上，以及它们的位置号。

**6、猴子吃桃问题**

猴子第一天摘下若干桃子，当即吃了一半，还不过瘾，又多吃了一个。第二天早上又将剩下的桃子吃掉的一半，又多吃了一个。以后每天早上都吃掉了前一天剩下的一半零一个。到第 10 天早上想再吃时，见只剩下一个桃子了。求第一天共摘多少桃子。（答案：1534）

# 第九章　回溯算法

搜索与回溯是计算机解题中常用的算法，很多问题无法根据某种确定的计算法则来求解，可以利用搜索与回溯的技术求解。回溯是搜索算法中的一种控制策略。它的基本思想是：为了求得问题的解，先选择某一种可能情况向前探索，在探索过程中，一旦发现原来的选择是错误的，就退回一步重新选择，继续向前探索，如此反复进行，直至得到解或证明无解。

如迷宫问题：进入迷宫后，先随意选择一个前进方向，一步步向前试探前进,如果碰到死胡同，说明前进方向已无路可走，这时，首先看其它方向是否还有路可走，如果有路可走，则沿该方向再向前试探；如果已无路可走，则返回一步，再看其它方向是否还有路可走；如果有路可走，则沿该方向再向前试探。按此原则不断搜索回溯再搜索，直到找到新的出路或从原路返回入口处无解为止。

## 递归回溯法算法框架[一]

```
procedure Try(k:integer);
  begin
    for i:=1 to 算符种数 Do
      if  满足条件 then
      begin
        保存结果
        if  到目的地 then 输出解
                    else Try(k+1);
        恢复：保存结果之前的状态{回溯一步}
      end;
    end;
```

## 递归回溯法算法框架[二]

```
procedure Try(k:integer);
  begin
    if  到目的地 then 输出解
       else
          for i:=1 to 算符种数 Do
            if  满足条件 then
                begin
                    保存结果
                    Try(k+1);
                end;
  end;
```

**例 1：素数环：** 把从 1 到 20 这 20 个数摆成一个环，要求相邻的两个数的和是一个素数。

**【算法分析】** 非常明显，这是一道回溯的题目。从 1 开始，每个空位有 20（19）种可能，只要填进去的数合法：与前面的数不相同；与左边相邻的数的和是一个素数。第 20 个数还要判断和第 1 个数的和是否素数。

〖算法流程〗1、数据初始化；　　2、递归填数：
判断第 J 种可能是否合法；
A、如果合法：填数；判断是否到达目标（20 个已填完）：是，打印结果；不是，递归填下一个；
B、如果不合法：选择下一种可能；
**【参考程序】**

```
program z74;框架[一]
var a:array[0..20]of byte;
    b:array[0..20]of boolean;
    total:integer;
function pd(x,y:byte):boolean;
 var k,i:byte;
 begin
   k:=2; i:=x+y; pd:=false;
   while (k<=trunc(sqrt(i)))and(i mod k<>0) do inc(k);
   if k>trunc(sqrt(i)) then pd:=true;
 end;
procedure print;
 var j:byte;
 begin
   inc(total);write('<',total,'>:');
   for j:=1 to 20 do write(a[j],' ');
   writeln;
 end;
procedure try(t:byte);
 var i:byte;
 begin
   for i:=1 to 20 do
   if pd(a[t-1],i)and b[i] then
   begin
     a[t]:=i;  b[i]:=false;
     if t=20 then begin if pd(a[20],a[1]) then print;end
             else try(t+1);
     b[i]:=true;
   end;
 end;
BEGIN
  fillchar(b,sizeof(b),#1);
  total:=0;
  try(1);
  write('total:',total);
END.
```

**例 2：设有 n 个整数的集合 {1,2,…,n}，从中取出任意 r 个数进行排列（r<n），试列出所有的排列。**

**解法一：**

**program it15; 框架[一]**

```
type se=set of 1..100;
```

```
VAR  s:se;n,r,num:integer;
     b:array [1..100] of integer;
PROCEDURE print;
 var i:integer;
  begin
   num:=num+1;
   for i:=1 to r do
     write(b[i]:3);
   writeln;
  end;
PROCEDURE try(k:integer);
  VAR  i:integer;
  begin
    for i:=1 to n do
       if i in s then
          begin
            b[k]:=i;
            s:=s-[i];
            if k=r then print
                   else try(k+1);
              s:=s+[i];
          end;
  end;
 BEGIN
   write('Input n,r:');readln(n,r);
   s:=[1..n];num:=0;
   try(1);
   writeln('number=',num);
 END.
```

**解法二：**

program it15; **框架[二]**

```
type se=set of 1..100;
VAR
   s:se;
   n,r,num,k:integer;
   b:array [1..100] of integer;

 PROCEDURE print;
 var i:integer;
  begin
   num:=num+1;
   for i:=1 to r do
     write(b[i]:3);
   writeln;
  end;
```

```
PROCEDURE try(s:se;k:integer);
 VAR i:integer;
 begin
    if k>r then print
        else
          for i:=1 to n do
             if i in s then
                begin
                  b[k]:=i;
                  try(s-[i],k+1);
                end;
 end;

BEGIN
  write('Input n,r:');
  readln(n,r);
  s:=[1..n];num:=0;
  try(s,1);
  writeln('number=',num);
  readln;
END.
```

**例 3、任何一个大于 1 的自然数 n,总可以拆分成若干个小于 n 的自然数之和.**

当 n=7 共 14 种拆分方法：

7=1+1+1+1+1+1+1
7=1+1+1+1+1+2
7=1+1+1+1+3
7=1+1+1+2+2
7=1+1+1+4
7=1+1+2+3
7=1+1+5
7=1+2+2+2
7=1+2+4
7=1+3+3
7=1+6
7=2+2+3
7=2+5
7=3+4
total=14

**解法一：**{参考程序}

```
program jjj;
var a:array[0..100]of integer;n,t,total:integer;
procedure print(t:integer);
var i:integer;
 begin
  write(n,'=');
  for i:=1 to t-1 do  write(a[i],'+');
  writeln(a[t]);
```

```
  total:=total+1;
 end;
procedure try(s,t:integer);
var i:integer;
  begin
    for i:=1 to s do
     if (a[t-1]<=i)and(i<n) then
      begin
        a[t]:=i;
        s:=s-a[t];
        if s=0 then print(t)
               else try(s,t+1);
        s:=s+a[t];
      end;
end;
begin
  readln(n);
  try(n,1);
  writeln('total=',total);
  readln;
end.
```

**解法二：**{参考程序}

```
program aaa;
var a:array[0..100]of integer;
    n,m:integer;
procedure try(x,y:integer);
var w,k:integer;
begin
  for k:=a[x-1] to n-1 do
  if y>=k then
  begin
    a[x]:=k;
    if y=k then
      begin
        write(n,'=');
        for w:=1 to x do write(a[w],'+');
        writeln(#8' '#8);
      end
    else try(x+1,y-k);
  end;
end;
begin
  readln(n);a[0]:=1;m:=n;
  try(1,m);
  readln;
end.
```

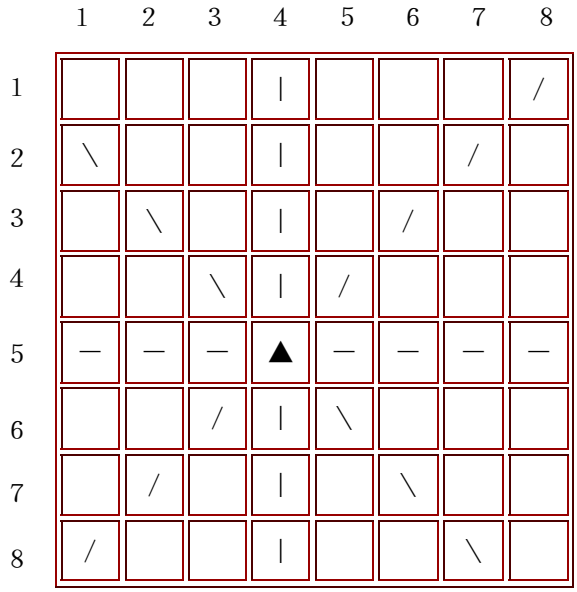
**例 4、八皇后问题：**要在国际象棋棋盘中放八个皇后，使任意两个皇后都不能互相吃。（提示：皇后能吃同一行、同一列、同一对角线的任意棋子。）

**放置第 i 个皇后的算法为：**

```
procedure Try(i);
  begin
    for 第 i 个皇后的位置=1 to 8 do;
      if 安全 then
        begin
          放置第 i 个皇后;
          对放置皇后的位置进行标记;
          if i=8 then 输出
              else Try(i+1); {放置第 i+1 个皇后}
          对放置皇后的位置释放标记，尝试下一个位置是否可行;
        end;
  end;
```

## 【算法分析】

显然问题的键在于如何判定某个皇后所在的行、列、斜线上是否有别的皇后；可以从矩阵的特点上找到规律，如果在同一行，则行号相同；如果在同一列上，则列号相同；如果同在 / 斜线上的行列值之和相同；如果同在 \ 斜线上的行列值之差相同；如果斜线不分方向，则同一斜线上两皇后的行号之差的绝对值与列号之差的绝对值相同。从下图可验证：




对于一组布局我们可以用一个一维数组来表示：A:ARRAY [1..8] OF INTEGER;A[I]的下标 I 表示第 I 个皇后在棋盘的第 I 行，A[I]的内容表示在第 I 行的第 A[I]列，例如：A[3]=5 就表示第 3 个皇后在第 3 行的第 5 列。在这种方式下，要表示两个皇后 I 和 J 不在同一列或斜线上的条件可以描述为：A[I]<>A[J] AND ABS(I-J)<>ABS(A[I]-A[J]){I 和 J 分别表示两个皇后的行号}

考虑每行有且仅有一个皇后，设一维数组A[1..8]表示皇后的放置：第i行皇后放在第j列，用A[i]＝j来表示，即下标是行数，内容是列数。

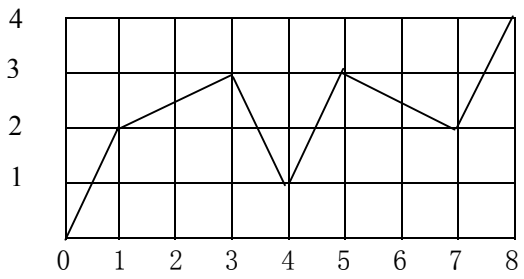
判断皇后是否安全，即检查同一列、同一对角线是否已有皇后，建立标志数组b[1..8]控制同一列只能有一个皇后，若两皇后在同一对角线上，则其行列坐标之和或行列坐标之差相等，故亦可建立标志数组c[1..16]、d[-7..7]控制同一对角线上只能有一个皇后。

从分析中，我们不难看出，搜索前进过程实际上是不断递归调用的过程，当递归返回时即为回溯的过程。

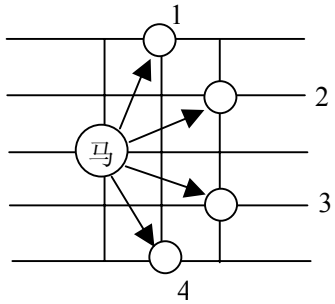
```
program ex1;
var  a:array[1..8] of byte;
     b:array[1..8] of boolean;
     c:array[1..16] of boolean;
     d:array[-7..7] of boolean;
     sum:byte;
procedure pr;
  var i:byte;
  begin
    for i:=1 to 8 do write(a[i]:4);
    inc(sum);writeln(' sum=',sum);
  end;
procedure try(t:byte);
  var j:byte;
  begin
    for j:=1 to 8 do{每个皇后都有8种可能位置}
      if b[j] and c[t+j] and d[t-j] then {寻找放置皇后的位置}
        begin  {放置皇后,建立相应标志值}
          a[t]:=j;{摆放皇后}
          b[j]:=false;{宣布占领第j列}
          c[t+j]:=false;{占领两个对角线}
          d[t-j]:=false;
          if t=8 then pr  {8个皇后都放置好,输出}
            else try(t+1);{继续递归放置下一个皇后}
          b[j]:=true; {递归返回即为回溯一步,当前皇后退出}
          c[t+j]:=true;
          d[t-j]:=true;
        end;
  end;
BEGIN
  fillchar(b,sizeof(b),#1);
  fillchar(c,sizeof(c),#1);
  fillchar(d,sizeof(d),#1);
  sum:=0;
  try(1);{从第1个皇后开始放置}
END.
```

## 例 5：马的遍历

中国象棋半张棋盘如图 4（a）所示。马自左下角往右上角跳。今规定只许往右跳，不许往左跳。比如图 4（a）中所示为一种跳行路线，并将所经路线打印出来。打印格式为：0，0->2,1->3,3->1,4->3,5->2,7->4,8…




（a）




（b）

图 4

分析：如图 4（b），马最多有四个方向，若原来的横坐标为 j、纵坐标为 i,则四个方向的移动可表示为：

1：　（i,j）→（i+2,j+1）；　(i<3,j<8)
2：　（i,j）→（i+1,j+2）；　(i<4,j<7)
3：　（i,j）→（i-1,j+2）；　(i>0,j<7)
4：　（i,j）→（i-2,j+1）；　(i>1,j<8)

搜索策略：

S1:A[1]:=(0,0);

S2:从A[1]出发，按移动规则依次选定某个方向，如果达到的是（4,8）则转向 S3,否则继续搜索下一个到达的顶点；

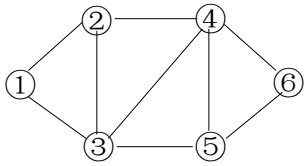
S3:打印路径。

```
program exam2;
const  x:array[1..4,1..2] of integer=((2,1),(1,2),(-1,2),(-2,1)); {四种移动规则}
var  t:integer; {路径总数}
     a:array[1..9,1..2] of integer;   {路径}

procedure print(ii:integer);  {打印}
var   i:integer;
begin
  inc(t);                    {路径总数}
  for i:=1 to ii-1 do
     write(a[i,1],',',a[i,2],'-->');
  writeln('4,8',t:5);
  readln;
end;

procedure try(i:integer);  {搜索}
var  j:integer;
```

```
begin
  for j:=1 to 4 do
    if (a[i-1,1]+x[j,1]>=0) and (a[i-1,1]+x[j,1]<=4) and
       (a[i-1,2]+x[j,2]>=0) and (a[i-1,2]+x[j,2]<=8) then
       begin
         a[i,1]:=a[i-1,1]+x[j,1];
         a[i,2]:=a[i-1,2]+x[j,2];
         if (a[i,1]=4) and (a[i,2]=8) then
            print(i)
         else try(i+1);  {搜索下一步}
         a[i,1]:=0;a[i,2]:=0
       end;
end;
BEGIN  {主程序}
  try(2);
END.
```

**【例 6】**设有一个连接 n 个地点①—⑥的道路网，找出从起点①出发到过终点⑥的一切路径，要求在每条路径上任一地点最多只能通过一次。

**【算法分析】**




从①出发，下一点可到达②或③，可以分支。具体步骤为：

⑴假定从起点出发数起第 k 个点 Path[k]，如果该点是终点 n 就打印一条路径；

⑵如果不是终点 n，且前方点是未曾走过的点，则走到前方点，定(k+1)点为到达路径，转步骤⑴；

(3)如果前方点已走过，就选另一分支点；

(4)如果前方点已选完，就回溯一步，选另一分支点为出发点；

(5)如果已回溯到起点，则结束。

为了表示各点的连通关系，建立如下的关系矩阵：

第一行表示与①相通点有②③，0 是结束标志;以后各行依此类推。

$$\text{Roadnet}=\begin{bmatrix} 2 & 3 & 0 & & \\ 1 & 3 & 4 & 0 & \\ 1 & 2 & 4 & 5 & 0 \\ 2 & 3 & 5 & 6 & 0 \\ 3 & 4 & 6 & 0 & \\ 4 & 5 & 0 & & \end{bmatrix}$$

集合 b 是为了检查不重复点。

```
Program Exam68;
  const n=6;
      roadnet: array[1..n, 1..n] of 0..n=( (2,3,0,0,0,0),
                                           (1,3,4,0,0,0),
                                           (1,2,4,5,0,0),
                                           (2,3,5,6,0,0),
                                           (3,4,6,0,0,0),
                                           (4,5,0,0,0,0) );
  var b: set of 1..n;
```

```
    path: array[1..n] of 1..n;
    p: byte;
procedure prn(k: byte);
 var i: byte;
 begin
    inc(p); write(' <' , p:2, ' >' , '  ' :4);
    write (path[1]:2);
    for I:=2 to k do
     write (' --' , path[ i ]:2);
    writeln
 end;
procedure try(k: byte);
 var j: byte;
 begin
    j:=1;
    repeat
     path[k]:=roadnet [path [k-1], j ];
     if not (path [k] in b) then
       begin
         b:=b+[path [k] ];
         if path [k]=n then prn (k)
                       else try(k+1);
         b:=b-[path [k] ];
       end;
     inc(j);
    until roadnet [path [k-1], j ]=0
 end;
BEGIN
  b:=[1]; p=0; path[1]:=1;
  try(2);
  readln
END.
```

## 【上机练习】

**例 1、**输出自然数 1 到 n 所有不重复的排列，即 n 的全排列。

［分析］求 N 个数的全排列，可以看成把 N 个不同的球放入 N 个不同的盒子中，每个盒子中只能有一个球。解法与八皇后问题相似。［参考过程］

```
procedure try(I:integer);
var j:integer;
 begin
      for j:=1 to n do
           if a[j]=0 then
          begin
               x[I]:=j;
               a[j]:=1;
               if I<n then try(I+1)
                  else print;
               a[j]=0;
          end;
 end;
```

**例 2、**找出 n 个自然数(1,2,3,…,n)中 r 个数的组合。例如，当 n=5,r=3 时，所有组合为：
输出格式

1 2 3
1 2 4
1 2 5
1 3 4
1 3 5
1 4 5
2 3 4
2 3 5
2 4 5
3 4 5
total=10 {组合的总数}

[分析:]设在 b[1],b[2],…,b[i-1]中已固定地取了某一组值且 b[i-1]=k 的前提下，过程 try(i,k)能够列出所有可能的组合。由于此时 b[i]的取值只能是 k+1 至 n-r+i,我可以对 j=k+1,k+2,…,n-r+i,使 b[i]:=j,再调用过程 try(i+1,j)。这就形成了递归调用。直至 i 的值大于 r 时，就可以在 b 中构成一种组合并输出。

**例 3、输出字母 a、b、c、d，4 个元素全排列的每一种排列。**

**例 4、**有 A、B、C、D、E 五本书，要分给张、王、刘、赵、钱五位同学，每人只能选一本，事先让每人把自己喜爱的书填于下表，编程找出让每人都满意的所有方案。

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 张 | | | Y | Y | |
| 王 | Y | Y | | | Y |
| 刘 | | Y | Y | | |
| 赵 | Y | Y | | Y | |
| 钱 | | Y | | | Y |

【答案】四种方案

| | 张 | 王 | 刘 | 赵 | 钱 |
|---|---|---|---|---|---|
| ① | C | A | B | D | E |
| ② | D | A | C | B | E |
| ③ | D | B | C | A | E |
| ④ | D | E | C | A | B |

**例 5、**有红球 4 个，白球 3 个，黄球 3 个，将它们排成一排共有多少种排法？

**【算法分析】**可以用回溯法来生成所有的排法。用数组 b[1..3]表示尚未排列的这 3 种颜色球的个数。设共有 I-1 个球已参加排列，用子程序 try(i)生成由第 I 个位置开始的以后 n-I+1 位置上的各种排列。对于第 I 个位置，我们对 3 种颜色的球逐一试探，看每种颜色是否还有未加入排序的球。若有，则选取一个放在第 I 个位置上，且将这种球所剩的个数减 1，然后调用 try(I+1),直至形成一种排列后出。对第 I 个位置上的所有颜色全部试探完后，则回溯至前一位置。

**例 6、**显示从前 m 个大写英文字母中取 n 个不同字母的所有种排列。

# 第十章　贪心算法

若在求解一个问题时，能根据每次所得到的局部最优解，推导出全局最优或最优目标。那么，我们可以根据这个策略，每次得到局部最优解答，逐步而推导出问题，这种策略称为贪心法。下面我们看一些简单例题。

**例 1：在 N 行 M 列的正整数矩阵中，要求从每行中选出 1 个数，使得选出的总共 N 个数的和最大。**

**【算法分析】**

要使总和最大，则每个数要尽可能大，自然应该选每行中最大的那个数。因此，我们设计出如下算法:

```
读入 N, M,矩阵数据;
Total := 0;
For I := 1 to N do begin          {对 N 行进行选择}
  选择第 I 行最大的数,记为 K;
   Total := Total + K;
End;
输出最大总和 Total;
```

从上例中我们可以看出，和递推法相仿，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，而是做一个局部的最优选择，即贪心选择（在例中，这种贪心选择表现为选择一行中的最大整数），这样，不断的将问题归纳为若干相似的子问题，最终产生出一个全局最优解。

特别注意的是是，局部贪心的选择是否可以得出全局最优是能否采用贪心法的关键所在。对于能否使用贪心策略，应从理论上予以证明。下面我们看看另一个问题。

**例 2：部分背包问题**

给定一个最大载重量为 M 的卡车和 N 种食品，有食盐，白糖，大米等。已知第 i 种食品的最多拥有 $W_i$ 公斤，其商品价值为 $V_i$ 元/公斤，编程确定一个装货方案，使得装入卡车中的所有物品总价值最大。

**【算法分析】**

因为每一个物品都可以分割成单位块，单位块的利益越大显然总收益越大，所以它局部最优满足全局最优，可以用贪心法解答，方法如下：先将单位块收益按从大到小进行排列，然后用循环从单位块收益最大的取起，直到不能取为止便得到了最优解。

因此我们非常容易设计出如下算法：

```
问题初始化;                    {读入数据}
按 Vi 从大到小将商品排序;
I := 1;
repeat
  if M = 0 then Break;   {如果卡车满载则跳出循环}
  M := M - Wi;
  if M >= 0 then 将第 I 种商品全部装入卡车
  else
    将(M + Wi)重量的物品 I 装入卡车;
  I := I + 1;                  {选择下一种商品}
until (M <= 0) OR (I >N)
```

在解决上述问题的过程中，首先根据题设条件，找到了贪心选择标准($V_i$)，并依据这个标准直接逐步去求最优解，这种解题策略被称为贪心法。

因此，利用贪心策略解题，需要解决两个问题：

首先，确定问题是否能用贪心策略求解；一般来说，适用于贪心策略求解的问题具有以下特点：

①可通过局部的贪心选择来达到问题的全局最优解。运用贪心策略解题，一般来说需要一步步的进行多次的贪心选择。在经过一次贪心选择之后，原问题将变成一个相似的，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，且每一个选择都仅做一次。

②原问题的最优解包含子问题的最优解，即问题具有最优子结构的性质。在背包问题中，第一次选择单位重量价值最大的货物，它是第一个子问题的最优解，第二次选择剩下的货物中单位重量价值最大的货物，同样是第二个子问题的最优解，依次类推。

③其次，如何选择一个贪心标准？正确的贪心标准可以得到问题的最优解，在确定采用贪心策略解决问题时，不能随意的判断贪心标准是否正确，尤其不要被表面上看似正确的贪心标准所迷惑。在得出贪心标准之后应给予严格的数学证明。

下面来看看 0-1 背包问题。

给定一个最大载重量为 M 的卡车和 N 种动物。已知第 i 种动物的重量为 $W_i$，其最大价值为 $V_i$，设定 M，$W_i$，$V_i$ 均为整数，编程确定一个装货方案，使得装入卡车中的所有动物总价值最大。

分析：对于 N 种动物，要么被装，要么不装，也就是说在满足卡车载重的条件下，如何选择动物，使得动物价值最大的问题。

即确定一组 X1，X2，…，Xn， $Xi \in \{0,1\}$

$f(x)=\max(\sum X_i*V_i)$ 其中，$\sum(X_i*W_i) \leqq W$

从直观上来看，我们可以按照上例一样选择那些价值大，而重量轻的动物。也就是可以按价值质量比（$V_i/W_i$）的大小来进行选择。可以看出，每做一次选择，都是从剩下的动物中选择那些 $V_i/W_i$ 最大的，这种局部最优的选择是否能满足全局最优呢？我们来看看一个简单的例子：

设 N=3，卡车最大载重量是 100，三种动物 A、B、C 的重量分别是 40，50，70，其对应的总价值分别是 80、100、150。

情况 A：按照上述思路，三种动物的 $V_i/W_i$ 分别为 2,2,2.14。显然，我们首先选择动物 C，得到价值 150，然后任意选择 A 或 B，由于卡车最大载重为 100，因此卡车不能装载其他动物。

情况 B：不按上述约束条件，直接选择 A 和 B。可以得到价值 80+100=180，卡车装载的重量为 40+50=90。没有超过卡车的实际载重，因此也是一种可行解，显然，这种解比上一种解要优化。

问题出现在什么地方呢？我们看看图 2-18

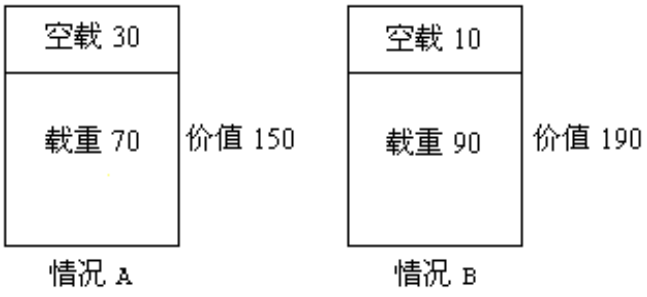



图 23 卡车装载货物情况分析

从图 23 中明显可以看出，情况 A，卡车的空载率比情况 B 高。也就是说，上面的分析，只考虑了货物的价值质量比，而没有考虑到卡车的运营效率，因此，局部的最优化，不能导致全局的最优化。

因此，贪心不能简单进行，而需要全面的考虑，最后得到证明。

**例 3:排队打水问题**

有 N 个人排队到 R 个水龙头去打水，他们装满水桶的时间为 T1，T2，…，Tn 为整数且各不相等，应如何安排他们的打水顺序才能使他们花费的时间最少？

分析：由于排队时，越靠前面的计算的次数越多，显然越小的排在越前面得出的结果越小（可以用数学方法简单证明，这里就不再赘述），所以这道题可以用贪心法解答，基本步骤：

(1)将输入的时间按从小到大排序；

(2)将排序后的时间按顺序依次放入每个水龙头的队列中；

(3)统计，输出答案。

**【样例输入】**

```
4  2          {4 人打水，2 个水龙头}
2  6  4  5    {每个打水时间}
```

**【样例输出】**

```
23            {总共花费时间}
```

**参考程序主要框架如下：**

```
Fillchar(S,Sizeof(S),0);
J:=0; Min:=0;
For I:=1 To N Do          {用贪心法求解}
  Begin
    Inc(J);
    If J=M+1 Then J:=1;
    S[J]:=S[J]+A[I];
    Min:=Min+S[J];
  End;
Writeln(Min);             {输出解答}
```

**例 4:均分纸牌（NOIP2002）**

有 N 堆纸牌，编号分别为 1，2，…, N。每堆上有若干张，但纸牌总数必为 N 的倍数。可以在任一堆上取若干张纸牌，然后移动。

移牌规则为：在编号为 1 堆上取的纸牌，只能移到编号为 2 的堆上；在编号为 N 的堆上取的纸牌，只能移到编号为 N-1 的堆上；其他堆上取的纸牌，可以移到相邻左边或右边的堆上。

现在要求找出一种移动方法，用最少的移动次数使每堆上纸牌数都一样多。

例如 N=4，4 堆纸牌数分别为：

① 9 ② 8 ③ 17 ④ 6

移动 3 次可达到目的：

从 ③ 取 4 张牌放到 ④ （9 8 13 10） -> 从 ③ 取 3 张牌放到 ②（9 11 10 10） -> 从 ② 取 1 张牌放到①（10 10 10 10）。

[输 入]：

键盘输入文件名。文件格式：

N（N 堆纸牌，1 <= N <= 100）

A1 A2 … An （N 堆纸牌，每堆纸牌初始数，1<= Ai <=10000）

[输 出]：

输出至屏幕。格式为：所有堆均达到相等时的最少移动次数。

**【样例输入】**(a.in)

4

9 8 17 6

**【样例输出】**(a.out)

3

**【算法分析】**

如果你想到把每堆牌的张数减去平均张数，题目就变成移动正数，加到负数中，使大家都变成 0，那就意味着成功了一半！拿例题来说，平均张数为 10,原张数 9，8，17，6，变为-1，-2，7，-4，其中没有为 0 的数，我们从左边出发：要使第 1 堆的牌数-1 变为 0，只须将-1 张牌移到它的右边（第 2 堆）-2 中；结果是-1 变为 0，-2 变为-3，各堆牌张数变为 0，-3，7，-4；同理：要使第 2 堆变为 0，只需将-3 移到它的右边（第 3 堆）中去，各堆牌张数变为 0，0，4，-4；要使第 3 堆变为 0，只需将第 3 堆中的 4 移到它的右边（第 4 堆）中去，结果为 0，0，0，0，完成任务。每移动 1 次牌，步数加 1。也许你要问，负数张牌怎么移，不违反题意吗？其实从第 i 堆移动-m 张牌到第 i+1 堆，等价于从第 i+1 堆移动 m 张牌到第 i 堆，步数是一样的。

如果张数中本来就有为 0 的，怎么办呢？如 0，-1，-5，6，还是从左算起（从右算起也完全一样），第 1 堆是 0，无需移牌，余下与上相同；再比如-1，-2，3，10，-4，-6，从左算起，第 1 次移动的结果为 0，-3，3，10，-4，-6；第 2 次移动的结果为 0，0，0，10，-4，-6，现在第 3 堆已经变为 0 了，可节省 1 步，余下继续。

**参考程序主要框架如下：**

```
ave:=0;step:=0;
for i:=1 to n do
begin
  read(f,a[i]); inc(ave,a[i]);      {读入各堆牌张数，求总张数 ave}
end;
ave:=ave div n;                     {求牌的平均张数 ave}
for i:=1 to n do a[i]:=a[i]-ave;    {每堆牌的张数减去平均数}
i:=1;j:=n;
while (a[i]=0) and (i<n) do inc(i);{过滤左边的 0}
while (a[j]=0) and (j>1) do dec(j);{过滤右边的 0}
while (i<j) do
begin
  inc(a[i+1],a[i]);                 {将第 i 堆牌移到第 i+1 堆中去}
  a[i]:=0;                          {第 i 堆牌移走后变为 0}
  inc(step);                        {移牌步数计数}
  inc(i);                           {对下一堆牌进行循环操作}
  while (a[i]=0) and (i<j) do inc(i);{过滤移牌过程中产生的 0}
end;
writeln(step);
```

点评：基本题（较易） 本题有 3 点比较关键：一是善于将每堆牌数减去平均数，简化了问题；二是要过滤掉 0（不是所有的 0，如-2，3，0，-1 中的 0 是不能过滤的）；三是负数张牌也可以移动，这是辩证法（关键中的关键）。

**例 5:删数问题：（NOI94）**

键盘输入一个高精度的正整数 N（此整数中没有‘0’），去掉其中任意 S 个数字后剩下的数字按原左右次序将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。

**【输入】**n

s

**【输出】**最后剩下的最小数。

**【样例输入】**

175438

4

**【样例输出】**

13

**【算法分析】**

由于正整数 n 的有效数位为 240 位，所以很自然地采用字符串类型存贮 n。那么如何决定哪 s 位被删除呢？是不是最大的 s 个数字呢？显然不是，大家很容易举出一些反例。为了尽可能逼近目标，我们选取的贪心策略为：每一步总是选择一个使剩下的数最小的数字删去，即按高位到低位的顺序搜索，若各位数字递增，则删除最后一个数字；否则删除第一个递减区间的首字符，这样删一位便形成了一个新数字串。然后回到串首，按上述规则再删下一个数字。重复以上过程 s 次为止，剩下的数字串便是问题的解了。

例如：n=178543

s=4

删数的过程如下：

n=178543 {删掉 8}

17543 {删掉 7}

1543 {删掉 5}

143 {删掉 4}

13 {解为 13}

这样，删数问题就与如何寻找递减区间首字符这样一个简单的问题对应起来。不过还要注意一个细节性的问题，就是可能会出现字符串串首有若干 0 的情况，甚至整个字符串都是 0 的情况。按以上贪心策略编制的程序框架如下：

```
输入 n，s；
while s>0 do
  begin
    i：=1;                                  {从串首开始找}
    while (i<length(n)) and (n[i]<=n[i+1]) do  i:=i+1;
    delete (n，i，1);                       {删除字符串 n 的第 i 个字符}
    s:=s-1;
  end;
while (length(n)>1)and (n[1]= ‘0’ ) do delete(n，1，1);
输出 n;                                     {删去串首可能产生的无用零}
```

**例 6：拦截导弹问题。（NOIP1999）**

某国为了防御敌国的导弹袭击，开发出一种导弹拦截系统，但是这种拦截系统有一个缺陷：虽然它的第一发炮弹能够到达任意的高度，但是以后每一发炮弹都不能高于前一发的高度。某天，雷达捕捉到敌国的导弹来袭，由于该系统还在试用阶段。所以一套系统有可能不能拦截所有的导弹。

输入导弹依次飞来的高度（雷达给出的高度不大于 30000 的正整数）。计算要拦截所有导弹最小需要配备多少套这种导弹拦截系统。

● 输入数据

导弹数 n 和 n 颗依次飞来的高度（1≤n≤1000）.

● 输出数据

要拦截所有导弹最小配备的系统数 k。

**【算法分析】**

按照题意，被一套系统拦截的所有导弹中，最后一枚导弹的高度最低。设：

k 为当前配备的系统数；

L[k]为被第 k 套系统拦截的最后一枚导弹的高度，简称系统 k 的最低高度（1≤k≤n）。

我们首先设导弹 1 被系统 1 所拦截(k←1,L[k]←导弹 1 的高度)。然后依次分析导弹 2，…，导弹 n 的高度。

若导弹 I 的高度高于所有系统的最低高度，则断定导弹 I 不能被这些系统所拦截，应增设一套系统来拦截导弹 I(k←k+1,L[k]←导弹 i 的高度)；若导弹 I 低于某些系统的最低高度，那么导弹 I 均可被这些系统所拦截。究竟选择哪个系统拦截可使得配备的系统数最少，我们不妨采用贪心策略，选择其中最低高度最小（即导弹 I 的高度与系统最低高度最接近）的一套系统 P(L[p]=min{L[j]|L[j]>导弹 I 的高度}；L[p]←导弹 I 的高度)（i≤j≤k）。这样可使得一套系统拦截的导弹数尽可能增多。

依次类推，直至分析了 n 枚导弹的高度为止。此时得出的 k 便为应配备的最少系统数。

**参考程序主要框架如下：**

```
k:=1;L[1]:=导弹 1 的高度;
for i:=2 to n do
  begin
    p:=0;
    for j:=1 to k do
       if (L[j]>=导弹 I 的高度)and((p=0)or(L[j]<L[p])) then p:=j;
    if p=0 then begin k:=k+1;L[k]:= 导弹 I 的高度; end
            else L[p]:= 导弹 I 的高度;
  end;
```

输出应配备的最少系统数 K。

## 【上机练习】

**1、部分背包问题**

输入：M=50　N=3

| Wi | Vi |
|---|---|
| 30 | 4 |
| 20 | 5 |
| 10 | 6 |

输出：

| Num | Weight | V |
|---|---|---|
| 3 | 10 | 60 |
| 2 | 20 | 100 |
| 1 | 20 | 80 |

total=240

**2、排队打水问题**

3、均分纸牌（**NOIP2002**）
4、删数问题（NOI94）
5、拦截导弹问题（**NOIP1999**）

输入：N=5　导弹高度：7　9　6　8　5
输出：导弹拦截系统 K=2
输入：N=3　导弹高度：4　3　2
输出：导弹拦截系统 K=1
输入：N=3　导弹高度：2　3　4
输出：导弹拦截系统 K=3

6、排队接水(water.pas)

【问题描述】

有 n 个人在一个水龙头前排队接水，假如每个人接水的时间为 $T_i$，请编程找出这 n 个人排队的一种顺序，使得 n 个人的平均等待时间最小。

【输入】

输入文件共两行，第一行为 n；第二行分别表示第 1 个人到第 n 个人每人的接水时间 $T_1$，$T_2$，…，$T_n$，每个数据之间有 1 个空格。

【输出】

输出文件有两行，第一行为一种排队顺序，即 1 到 n 的一种排列；第二行为这种排列方案下的平均等待时间(输出结果精确到小数点后两位)。

【样例】

| water.in | water.out |
|---|---|
| 10 | 3 2 7 8 1 4 9 6 10 5 |
| 56 12 1 99 1000 234 33 55 99 812 | 291.90 |

7、智力大冲浪（riddle.pas）

【问题描述】

小伟报名参加中央电视台的智力大冲浪节目。本次挑战赛吸引了众多参赛者，主持人为了表彰大家的勇气，先奖励每个参赛者 m 元。先不要太高兴！因为这些钱还不一定都是你的？！接下来主持人宣布了比赛规则：

首先，比赛时间分为 n 个时段(n≤500)，它又给出了很多小游戏，每个小游戏都必须在规定期限 ti 前完成(1≤ti≤n)。如果一个游戏没能在规定期限前完成，则要从奖励费 m 元中扣去一部分钱 wi，wi 为自然数，不同的游戏扣去的钱是不一样的。当然，每个游戏本身都很简单，保证每个参赛者都能在一个时段内完成，而且都必须从整时段开始。主持人只是想考考每个参赛者如何安排组织自己做游戏的顺序。作为参赛者，小伟很想赢得冠军，当然更想赢取最多的钱！注意：比赛绝对不会让参赛者赔钱！

【输入】

输入文件 riddle.in，共 4 行。
第 1 行为 m，表示一开始奖励给每位参赛者的钱；
第 2 行为 n，表示有 n 个小游戏；
第 3 行有 n 个数，分别表示游戏 1 到 n 的规定完成期限；
第 4 行有 n 个数，分别表示游戏 1 到 n 不能在规定期限前完成的扣款数。

【输出】

输出文件 riddle.out，仅 1 行。表示小伟能赢取最多的钱。

【样例】

| riddle.in | riddle.out |
|---|---|
| 10000<br>7<br>4 2 4 3 1 4 6<br>70 60 50 40 30 20 10 | 9950 |

# 第十一章　分治算法策略

所谓分治就是指的分而治之，即将较大规模的问题分解成几个较小规模的问题，通过对较小规模问题的求解达到对整个问题的求解。当我们将问题分解成两个较小问题求解时的分治方法称之为二分法。

分治的基本思想是将一个规模为 n 的问题分解为 k 个规模较小的子问题，这些子问题互相独立且与原问题相同。找出各部分的解，然后把各部分的解组合成整个问题的解。

1、解决算法实现的同时，需要估算算法实现所需时间。分治算法时间是这样确定的：

解决子问题所需的工作总量（由子问题的个数、解决每个子问题的工作量 决定）合并所有子问题所需的工作量

2、分治法是把任意大小问题尽可能地等分成两个子问题的递归算法

3、分治的具体过程：

```
begin    {开始}
  if ①问题不可分 then ②返回问题解
    else begin
            ③从原问题中划出含一半运算对象的子问题 1;
            ④递归调用分治法过程，求出解 1;
            ⑤从原问题中划出含另一半运算对象的子问题 2;
            ⑥递归调用分治法过程，求出解 2;
            ⑦将解 1、解 2 组合成整修问题的解;
         end;
end; {结束}
```

## 二、例题分析

### 快速排序(递归算法)

**（1）基本思想：**

在当前无序区 R[1..H]中任取一个数据元素作为比较的"基准"(不妨记为 X)，用此基准将当前无序区划分为左右两个较小的无序区：R[1..I-1]和 R[I+1..H]，且左边的无序子区中数据元素均小于等于基准元素，右边的无序子区中数据元素均大于等于基准元素，而基准 X 则位于最终排序的位置上，即 R[1..I-1]≤X.Key≤R[I+1..H](1≤I≤H)，当 R[1..I-1]和 R[I+1..H]均非空时，分别对它们进行上述的划分过程，直至所有无序子区中的数据元素均已排序为止。

**（2）排序过程：**

【示例】：

| | |
|---|---|
| 初 始 关键字 | [49 38 65 97 76 13 27 49] |
| 第一次交换后 | [27 38 65 97 76 13 49 49] |
| 第二次交换后 | [27 38 49 97 76 13 65 49] |
| J 向左扫描，位置不变，第三次交换后 | [27 38 13 97 76 49 65 49] |
| I 向右扫描，位置不变，第四次交换后 | [27 38 13 49 76 97 65 49] |
| J 向左扫描 | [27 38 13 49 76 97 65 49] |

**（一次划分过程）**

初 始 关键字　[49　38　65　97　76　13　27　49]
一趟排序之后　[27　38　13] 49 [76　97　65　49]
二趟排序之后　[13] 27 [38] 49 [49　65] 76 [97]
三趟排序之后　13　27　38　49　49 [65] 76　97
最后的排序结果 13　27　38　49　49　65　76　97

**各趟排序之后的状态**

**快速排序算法 qsort**

```
Const     n=10;
Var    a:array [1..n] of integer;  k:integer;
procedure qsort(low,high:integer);
 var  i,j:integer;x:integer;
 begin
   if low<high then
     begin
       i:=low;j:=high;x:=a[i];
       repeat
         while (a[j]>=x)and(i<j)do j:=j-1;  {把大于基准的数留在右面}
         if (a[j]<x)and(i<j) then
           begin
             a[i]:=a[j];i:=i+1;            {把小于基准的数交换到左面}
           end;
         while (a[i]<=x)and (i<j) do i:=i+1;{把小于基准的数留在左面}
         if (a[i]>x)and(i<j) then
           begin
             a[j]:=a[i];j:=j-1;            {把大于基准的数交换到右面}
           end;
       until i=j;                  {直到 I 与 j 重叠，完成一次分组过程}
       a[i]:=x;            {把基准数插入相应的位置，以后不再参与排序}
       qsort(low,i-1);                          {小于基准的数重复分组}
       qsort(i+1,high);                         {大于基准的数重复分组}
     end;
   end;
```

**【例 2】用递归算法实现二分查找即：有 20 个已经从小到大排序好的数据，从键盘输入一个数 X，用对半查找方法，判断它是否在这 20 个数中。**

```
program ex0202_7;
var a:array[1..20]of integer;
    n,i,m,x,y:integer;
      procedure  jc(x,y:integer);
        var  k:integer;
            begin
               k:=(x+y)div 2;
               if a[k]=m then writeln('then num in',k:5);
               if x>y then writeln('no find')
               else begin
```

```
                    if a[k]<m then jc(k+1,y);
                    if a[k]>m then jc(x,k-1);
                  end;
          end;
begin
  readln(n);
  x:=1;y:=n;
  for i:=1to n do  readln(a[i]);
  readln(m);
  jc(x,y);
  readln;
end.
```

**【例 3】一元三次方程求解**

有形如：$ax^3+bx^2+cx+d=0$ 这样的一个一元三次方程。给出该方程中各项的系数（a，b，c，d 均为实数），并约定该方程存在三个不同实根（根的范围在—100 至 100 之间），且根与根之差的绝对值≥1。

要求由小到大依次在同一行输出这三个实根（根与根之间留有空格），并精确到小数点后 2 位。

提示：记方程 f（x）=0，若存在 2 个数 x1 和 x2，且 x1<x2，f（x1）*f（x2）＜0，则在（x1，x2）之间一定有一个根。

**输入：**

a，b，c，d

**输出：**

三个实根（根与根之间留有空格）

**输入输出样例**

输入：　1 -5 -4 20

输出：　-2.00 2.00 5.00

**【算法分析】**

这是一道有趣的解方程题。为了便于求解，将原方程

$$f(x)=ax^3+bx^2+cx+d=0$$

变换成

$$f'(x)=x^3+b'*x^2+c'*x+d'=0$$

的形式（其中 $b'=\left\lfloor\frac{b}{2}\right\rfloor$，$c'=\left\lfloor\frac{c}{2}\right\rfloor$，$d'=\left\lfloor\frac{d}{2}\right\rfloor$），f(x)和 f'(x)的根不变。设 x 的值域（-100 至 100 之间）中有 x， 其左右两边相距 0.0005 的地方有 $x_1$ 和 $x_2$ 两个数，即 $x_1=x-0.0005$ $x_2=x+0.0005$

$x_1$ 和 $x_2$ 间的距离（0.001）满足精度要求（精确到小数点后 2 位）。若出现如图 1 所示的两种情况之一，则确定 x 为 f'(x)=0 的根。

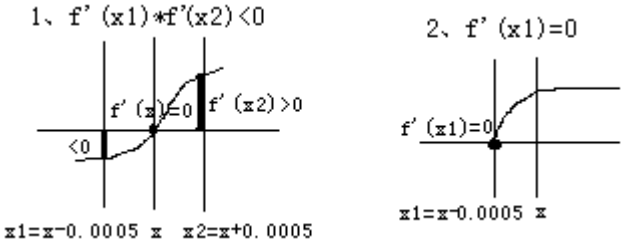



图 1

有两种方法计算 f’(x)=0 的根 x:

**1．枚举法**

根据根的值域和根与根之间的间距要求(≥1),我们不妨将根的值域扩大 100 倍(-10000≤x≤10000），依次枚举该区间的每一个整数值 x，并在题目要求的精度内设定区间：$x_1=\frac{x-0.05}{100}$，$x_2=\frac{x+0.05}{100}$。若区间端点的函数值 f’($x_1$)和 f’($x_2$)异号或者在区间端点 $x_1$ 的函数值 f’($x_1$)=0，则确定 $\frac{x}{100}$ 为 f’(x)=0 的一个根。

由此得出算法：

```
输入方程中各项的系数 a，b，c，d ；
b←b/a;  c←c/a;  d←d/a;  a←1;{将方程变换为 x^3+b'*x^2+c'*x+d'=0 的形式}
for x←-10000 to 10000 do                    {枚举当前根*100 的可能范围}
  begin
     x1←(x-0.05)/100;  x2←(x+0.05)/100;      {在题目要求的精度内设定区间}
     if (f(x1)*f(x2)<0) or (f(x1)=0)
          {若在区间两端的函数值异号或在 x1 处的函数值为 0，则确定 x/100 为根}
     then write(x/100:0:2,'' );
end; {for}
```

其中函数 f(x)计算 $x^3+b*x^2+c*x+d$：

```
function f(x:extended):extended;                {计算 x^3+b*x^2+c*x+d}
begin
    f←x*x*x+b*x*x+c*x+d;
end; {f}
```

**2.分治法**

枚举根的值域中的每一个整数 x(-100≤x≤100)。由于根与根之差的绝对值≥1，因此设定搜索区间[$x_1$，$x_2$]，其中 $x_1$=x，$x_2$=x+1。若

⑴f’($x_1$)=0，则确定 $x_1$ 为 f’(x)的根；

⑵f’($x_1$)*f’($x_2$)>0，则确定根 x 不在区间[$x_1$，$x_2$]内，设定[$x_2$，$x_2$+1]为下一个搜索区间

⑶f’($x_1$)*f’($x_2$)<0，则确定根 x 在区间[$x_1$，$x_2$]内。

如果确定根 x 在区间[$x_1$，$x_2$]内的话（f’($x_1$)*f’($x_2$)<0），如何在该区间找到根的确切位置。采用二分法，将区间[$x_1$，$x_2$]分成左右两个子区间：左子区间[$x_1$，x]和右子区间[x，$x_2$]（其中 x=$\frac{x_1+x_2}{2}$）：

如果 f’($x_1$)*f’(x)≤0，则确定根在左区间[$x_1$，x]内，将 x 设为该区间的右指针（$x_2$=x），继续对左区间进行对分；如果 f’($x_1$)*f’(x)>0，则确定根在右区间[x，$x_2$]内，将 x 设为该区间的左指针（$x_1$=x），继续对右区间进行对分；

上述对分过程一直进行到区间的间距满足精度要求为止（$x_2$-$x_1$<0.001）。此时确定 $x_1$ 为 f’(x)的根。

由此得出算法：

```
输入方程中各项的系数 a，b，c，d ；
b←b/a;  c←c/a;  d←d/a; a←1; {将方程变换为 x^3+b’*x^2+c’*x+d’=0 的形式}
for x←-100 to 100 do                                  {枚举每一个可能的根}
  begin
      x1←x; x2←x+1;                                   {确定根的可能区间}
      if f(x1)=0  then  write(x1:0:2,’ ’)             {若 x1 为根，则输出}
    else if (f(x1)*f(x2)<0)                           {若根在区间[x1，x2]中}
          then begin
                 while x2-x1>=0.001 do {若区间[x1，x2]不满足精度要求，则循环}
                  begin
                     xx←(x2+x1)/2;          {计算区间[x1，x2]的中间位置}
                     if f(x1)*f(xx)<=0      {若根在左区间，则调整右指针}
                     then  x2←xx  els e x1←xx; {若根在右区间，则调整左指针}
                   end; {while}
                   write(x1:0:2,’ ’); {区间[x1，x2]满足精度要求，确定 x1 为根}
                  end; {then}
  end; {for}
```

其中 f(x)的函数说明如枚举法所示。

## 【例 4】小车问题（car）

**【问题描述】**

甲、乙两人同时从 A 地出发要尽快同时赶到 B 地。出发时 A 地有一辆小车，可是这辆小车除了驾驶员外只能带一人。已知甲、乙两人的步行速度一样，且小于车的速度。问：怎样利用小车才能使两人尽快同时到达。

**【问题输入】**

仅一行，三个数据分别表示 AB 两地的距离 s，人的步行速度 a，车的速度 b。

**【问题输出】**

两人同时到达 B 地需要的最短时间。

**【输入输出样例】**

car.in

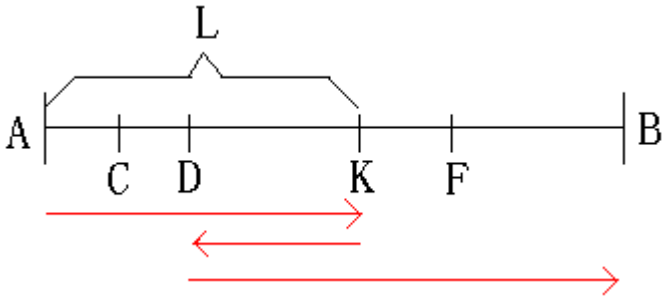
120　5　25

car.out

9.6000000000E+00

**【算法分析】**

最佳方案为：甲先乘车到达 K 处后下车步行，小车再回头接已走到 C 处的乙，在 D 处相遇后，乙再乘车赶往 B，最后甲、乙一起到达 B 地。如下图所示，这时所用的时间最短。这样问题就转换成了求 K 处的位置，我们用二分法，不断尝试，直到满足同时到达的时间精度。




算法框架如下：

1、输入 s，a，b；

2、c0:=0；c1:=s；c:=(c0+c1)/2；

3、求 t1，t2；

4、如果 t1<t2，那么 c:=(c0+c)/2

　　否则 c:=(c+c1)/2；

反复执行 3 和 4，直到 abs(t1-t2)满足精度要求（即小于误差标准）。

**【参考程序】**

```
program car1(input,output);
const zero=1e-4;
var s,a,b,c,c0,c1,t1,t2,t3,t4:real;
BEGIN
  assign(input,' car.in');
  assign(output,' car.out');
  reset(input);
  rewrite(output);
  readln(s,a,b);
  c0:=0;
  c1:=s;
  repeat
    c:=(c0+c1)/2;
    t3:=c/b;
    t1:=t3+(s-c)/a;
    t4:=(c-t3*a)/(a+b);
    t2:=t3+t4+(s-(t3+t4)*a)/b;
    if t1<t2 then c1:=c else c0:=c;
  until abs(t1-t2)<zero;
  writeln(t1);
  close(input);
  close(output)
END.
```

**【深入思考】**

现在把上述问题稍改一下，已知 A、B 两地相距 S=100 公里，在 A 地有 n 人，现有一辆汽车，此汽车除司机外只能载 1 人，已知汽车的速度为 $V_1$=50 公里/小时，人的速度为 $V_2$=5 公里/小时。要求设计一种方案，使得最后一个人用最少的时间到达 B。

## 【上机练习】

1、用非递归方法编写【例 2】

2、求 N 个数 A1，A2，A3……AN 中的最大值 MAX 和最小值 MIN。

【算法分析】

**策略一：**把 n(n>2)个数据先分成两组，分别求最大值、最小值，然后分别将两组的最大值和最小值进行比较，求出全部元素的最大值和最小值。若分组后元素还大于 2，则再次分组，直至组内元素小于等于 2。

**策略二：**如果 N 等于 1 时，MAX=MIN=A1，如果 N=2 时，MAX=A1，MIN=A2 或 MAX=A2，MIN=A1，这是非常简单的，所以此题可把所有的数作为一个序列，每次从中取出开头两个数，求共 MAX，MIN，然后再从剩余的数中取开头两个数，求其 MAX、MIN 后与前一次的 MAX、MIN 比较，可得出新的 MAX、MIN，这样重复下去，直到把所有的数取完（注意最后一次取可能是只有一个数），MAX，MIN 也就得到了。这就是典型的分治策略。**注意：这样做与把所有数字排序后再取 MAX、MIN 要快得多。**

3、求一元三次方程的解。

有形如：ax3+bx2+cx+d=0 一元三次方程。给出该方程中各项的系数（a，b，c，d 均为实数），并约定该方程存在三个不同实根（根的范围在 -100 至 100 之间），且根与根之差的绝对值 >=1。要求由小到大依次在同一行上输出这三个实根。

**输入：**a，b，c，d

**输出：** 三个实根（根与根之间留有空格）

输入输出样例

输入：1 －5 －4 20

输出：-2.00 2.00 5.00

4、在 n 个不同元素中找到第 k 小的元素。

【算法分析】

把这 n 个元素放在一个数组中，然后取出第 k 个元素为标准 m，把 n 个元素重新排列：小于标准 m 的元素放在数组前面，大于该标准的放在数组的后面。把该元素 m 放在两者之间。设小于标准的元素个数为 j-1，如果 j=k，则 A(k)即为所求元素。如果 j>k，则第 k 个元素必在区间[1,j]，因此取 A[1],…,A[j]为新的元素集合，然后重复上述的“部分排序”的过程。如果 j<k，则第 k 个元素必在区间[j+1,n]，因此取 A[j],…,A[n]为新的元素集合，重复该过程。直至 j=k 为止。

5、计算 $X^N$。

【算法分析】

**分析：**如果 N 为偶数，则 $X^N=(X^2)^{N/2}$；如果 N 为奇数，则 $X^N=X^{2P}*X^1$，（2P+1=N）转化成了偶数次方。这样递归地做下去，直到计算 $X^0$ 为止。

# 第十二章　　深度优先搜索算法

## 一、深度优先搜索的过程

深度优先搜索所遵循的搜索策略是尽可能“深”地搜索图。在深度优先搜索中，对于最新发现的节点，如果它还有以此为起点而未搜索的边，就沿此边继续搜索下去。当节点 v 的所有边都已被探寻过，搜索将回溯到发现节点 v 有那条边的始节点。这一过程一直进行到已发现从源节点可达的所有节点为止。如果还存在未被发现的节点，则选择其中一个作为源节点并重复以上过程，整个进程反复进行直到所有节点都被发现为止。即

1.以给定的某个顶点 V0 为起始点，访问该顶点；

2.选取一个与顶点 V0 相邻接且未被访问过的顶点 V1，用 V1 作为新的起始点，重复上述过程；

3.当到达一个其所有邻接的顶点都已被访问过的顶点 Vi 时，就退回到新近被访问过的顶点 Vi-1，继续访问 Vi-1 尚未访问的邻接点，重复上述搜索过程；

4.直到从任意一个已访问过的顶点出发，再也找不到未被访问过的顶点为止，遍历便告完成。

这种搜索的次序体现了向纵深发展的趋势，所以称之为深度优先搜索。

## 二、深度优先搜索特点

1、 由于深度搜索过程中有保留已扩展节点，则不致于重复构造不必要的子树系统。

2、 深度优先搜索并不是以最快的方式搜索到解，因为若目标节点在第 i 层的某处，必须等到该节点左边所有子树系统搜索完毕之后，才会访问到该节点，因此，搜索效率还取决于目标节点在解答树中的位置。

3、 由于要存储所有已被扩展节点，所以需要的内存空间往往比较大。

4、 深度优先搜索所求得的是仅仅是目前第一条从起点至目标节点的树枝路径，而不是所有通向目标节点的树枝节点的路径中最短的路径。

5、 适用范围：适用于求解一条从初始节点至目标节点的可能路径的试题。若要存储所有解答路径，可以再建立其它空间，用来存储每个已求得的解。若要求得最优解，必须记下达到目前目标的路径和相应的路程值，并与前面已记录的值进行比较，保留其中最优解，等全部搜索完成后，把保留的最优解输出。

## 三、深度优先搜索算法描述：

程序实现有两种方式—递归与非递归。

**1、递归**

递归过程为：

```
Procedure DEF-GO(step)
  for i:=1 to max do
    if 子结点符合条件 then
      产生新的子结点入栈；
        if 子结点是目标结点 then 输出
          else DEF-GO(step+1);
        栈顶结点出栈；
      endif;
    enddo;
```

**主程序为：**

```
Program DFS;
  初始状态入栈;
  DEF-GO(1);
```

**例如**　八数码难题--已知 8 个数的起始状态如图 1（a），要得到目标状态为图 1(b)。

| | | |
|---|---|---|
| 2 | 8 | 3 |
| 1 | 6 | 4 |
| 7 | ■ | 5 |

（a）

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 8 | ■ | 4 |
| 7 | 6 | 5 |

（b）

图　1

**【算法分析】**

求解时,首先要生成一棵结点的搜索树，按照深度优先搜索算法，我们可以生成图 2 的搜索树。图中，所有结点都用相应的数据库来标记，并按照结点扩展的顺序加以编号。其中，我们设置深度界限为 5 。粗线条路径表示求得的一个解。从图中可见，深度优先搜索过程是沿着一条路径进行下去，直到深度界限为止，回溯一步，再继续往下搜索，直到找到目标状态或 OPEN 表为空为止。

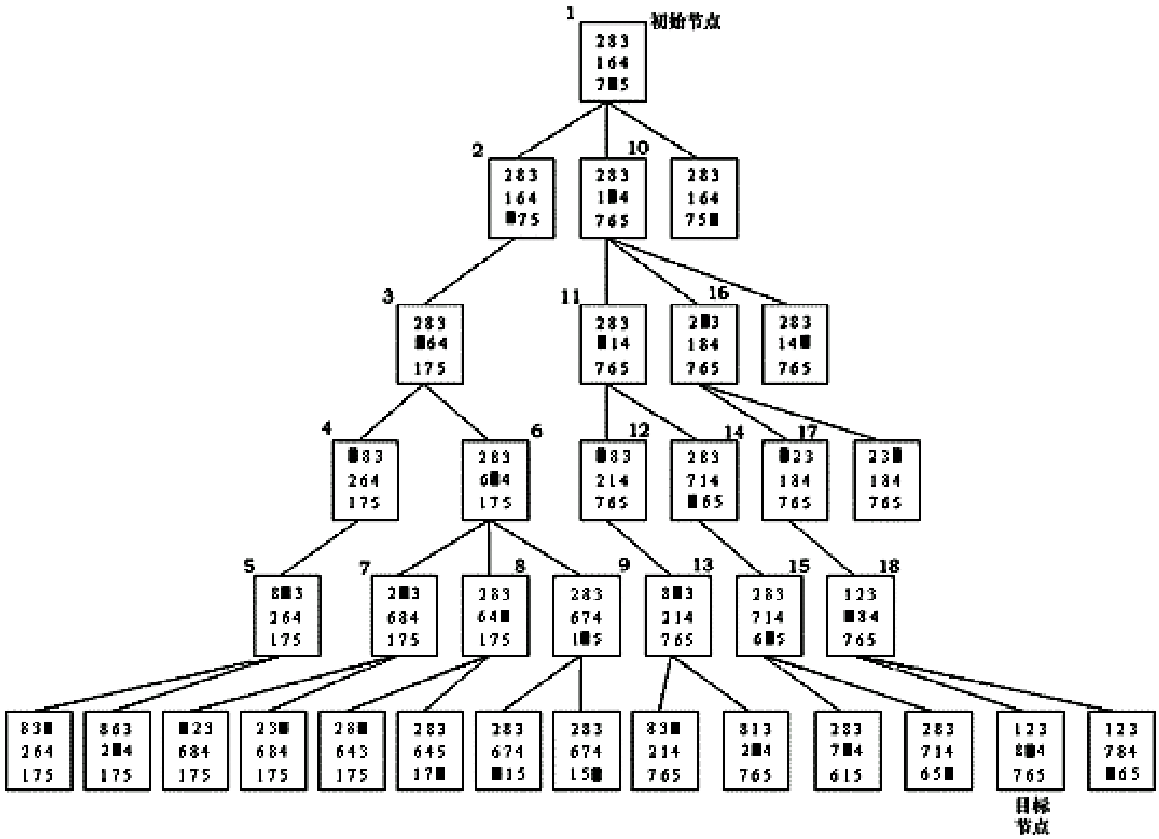



图 2

## 四、关于深度优先搜索的下界

对于许多问题，深度优先搜索查找的解答树可能含有无穷分支（深度优先搜索误入无穷分支就不可能找到目标节点），或者其深度可能至少要比某个可以接受的解答系列的已知上限还要深，或者能估计出目标节点不会超过多少层。为了避免可能太长的路径，给出一个节点扩展的最大深度，即深度界限 D，任何节点达到了 D，那么都将它们作为没有后继节点处理。如图 2 我们设置深度界限为 5，如果我们不对它的深度进行限定，那么第 5 层以下可以产生大量的搜索节点，而目标节点可以在第 5 层找到。深度优先搜索是最常用的算法之一，而确定“深度 D”是解题的关键，因为我们需要它消除不必要的搜索，提高搜索效率。

估算“深度 D”的方法：无章可循，凭经验和大致的计算，在时间和空间允许的范围内，宁大勿小。

**【例 1】**设有 A，B，C，D，E 五人从事 J1，J2，J3，J4，J5 五项工作，每人只能从事一项，他们的效益如下， 求最佳安排使效益最高。

| | J1 | J2 | J3 | J4 | J5 |
|---|---|---|---|---|---|
| A | 13 | 11 | 10 | 4 | 7 |
| B | 13 | 10 | 10 | 8 | 5 |
| C | 5 | 9 | 7 | 7 | 4 |
| D | 15 | 12 | 10 | 11 | 5 |
| F | 10 | 11 | 8 | 8 | 4 |

图 3

**分析:**每人选择五项工作中的一项，在各种选择的组合中，找到效益最高的的一种组合输出。

**【算法分析】**

1.数据库：用数组 f 构成堆栈存放产生的结点；数组 g 存放当前最高效益结点的组合；数组 p 作为结点是否选择过的标志位。

2.结点的产生：

(1)选择 p (i)=0 的结点；

(2)判断效益是否高于 g 记录结点的效益，是高于则更新 g 数组及最高效益值。

3.搜索策略: 深度优先搜索。

**【参考程序】**

```
program exam1;
const
  data: array [1..5,1..5] of integer
      =((13,11,10,4,7),(13,10,10,8,5),(5,9,7,7,4),
        (15,12,10,11,5),(10,11,8,8,4));
var
  i,max: integer;
  f,g: array [1..5] of integer;
  p: array [1..5] of integer;

procedure go(step,t: integer);  {选择最佳效益结点的组合}
var  i: integer;
begin
```

```
    for i:=1 to 5 do
      if p[i]=0 then begin
        f[step]:=i;
        p[i]:=1;
        t:=t+data[step,i];
        if step<5 then go(step+1,t)
        else
        if t>max then begin
          max:=t;
          g:=f;
        end;
        t:=t-data[step,i];
        p[i]:=0;
      end;
  end;
  begin
    max:=0;
    for i:=1 to 5 do p[i]:=0;
    go(1,0);
    writeln;
    for i:=1 to 5 do write(chr(64+i),':J',g[i],'':3);
    writeln;
    writeln('supply: ',max);
  end.
```

**【例 2】选书**

学校放寒假时，信息学竞赛辅导老师有 A，B，C，D，E 五本书，要分给参加培训的张、王、刘、孙、李五位同学，每人只能选一本书。老师事先让每个人将自己喜欢的书填写在如下的表格中。然后根据他们填写的表来分配书本，希望设计一个程序帮助老师求出所有可能的分配方案，使每个学生都满意。

| 学生 | A | B | C | D | E |
|---|---|---|---|---|---|
| 张同学 | | | Y | Y | |
| 王同学 | Y | Y | | | Y |
| 刘同学 | | Y | Y | | |
| 孙同学 | | | | Y | |
| 李同学 | | Y | | | Y |

**【算法分析】**

可用穷举法，先不考虑“每人都满意” 这一条件，这样只剩“每人选一本且只能选一本”这一条件。在这个条件下，可行解就是五本书的所有全排列，一共有 5！=120 种。然后在 120 种可行解中一一删去不符合“每人都满意”的解，留下的就是本题的解答。

为了编程方便，设 1，2，3，4，5 分别表示这五本书。这五个数的一种全排列就是五本书的一种分发。例如 54321 就表示第 5 本书（即 E）分给张，第 4 本书（即 D）分给王，……，

第 1 本书（即 A）分给李。“喜爱书表”可以用二维数组来表示，1 表示喜爱，0 表示不喜爱。

**算法设计：** S1：产生 5 个数字的一个全排列；

S2：检查是否符合“喜爱书表”的条件，如果符合就打印出来；

S3：检查是否所有的排列都产生了，如果没有产生完，则返回 S1；

S4：结束。

上述算法有可以改进的地方。比如产生了一个全排列 12345，从表中可以看出，选第一本书即给张同学的书，1 是不可能的，因为张只喜欢第 3、4 本书。这就是说，1××××一类的分法都不符合条件。由此想到，如果选定第一本书后，就立即检查一下是否符合条件，发现 1 是不符合的，后面的四个数字就不必选了，这样就减少了运算量。换句话说，第一个数字只在 3、4 中选择，这样就可以减少 3/5 的运算量。同理，选定了第一个数字后，也不应该把其他 4 个数字一次选定，而是选择了第二个数字后，就立即检查是否符合条件。例如，第一个数选 3，第二个数选 4 后，立即检查，发现不符合条件，就应另选第二个数。这样就把 34×××一类的分法在产生前就删去了。又减少了一部分运算量。

综上所述，改进后的算法应该是：在产生排列时，每增加一个数，就检查该数是否符合条件，不符合，就立刻换一个，符合条件后，再产生下一个数。因为从第 I 本书到第 I+1 本书的寻找过程是相同的，所以可以用递归方法。算法设计如下：

```
procedure try(i);
begin
   for j:=1 to 5 do
     begin
        if 第 i 个同学分给第 j 本书符合条件 then
          begin
             记录第 i 个数
             if i =5 then 打印一个解
                    else try(i+1);
                删去第 i 个数字
            end;
        end;
end;
```

**【参考程序】**

```
program exam3;
type five=1..5;
const
      like:array[five,five] of 0..1=((0,0,1,1,0),(1,1,0,0,1),
                          (0,1,1,0,0),(0,0,0,1,0),(0,1,0,0,1));
      name:array[five] of string[6]=('zhang','wang','liu','sun','li');
var  book:array[1..5] of 0..5;
     flag:set of five;c:integer;
procedure print;
var i:integer;
begin
  inc(c);writeln('answer',c,':');
  for i:=1 to 5 do
```

```
      writeln(name[i]:10,':',chr(64+book[i]));
end;
procedure try(i:integer);
var j:integer;
begin
  for j:=1 to 5 do
     if not(j in flag) and (like[i,j]>0) then
     begin
        flag:=flag+[j];book[i]:=j;
        if i=5 then print
               else try(i+1);
        flag:=flag-[j];book[i]:=0;
     end;
end;
begin
  flag:=[];c:=0;try(1);
  readln
end.
```

**输出结果：**

```
zhang: C
wang: A
liu: B
sun: D
li: E
```

**【例 3】跳马问题。**在 5*5 格的棋盘上，有一个国家象棋的马，从（1,1）点出发，按日字跳马，它可以朝 8 个方向跳，但不允许出界或跳到已跳过的格子上，要求在跳遍整个棋盘后再条回出发点。

输出前 5 个方案及总方案数。

输出格式示例：

```
1   16   21   10   25
20  11   24   15   22
17  2    19   6    9
12  7    4    23   14
3   18   13   8    5
```

{参考程序及注解}

```
var
a:array[1..5,1..5] of integer;    {记每一步走在棋盘的哪一格}
b:array[1..5,1..5] of boolean;    {棋盘的每一格有没有被走过}
u,v:array[1..8] of integer;       {8 个方向上的 x,y 增量}
i,j,num:integer;
procedure print;                  {打印方案}
  var  k,kk:integer;
```

```
begin
    num:=num+1;             {统计总方案}
    if num<=5 then          {打印出前 5 种方案}
    begin
       for k:=1 to 5 do     {打印本次方案}
          begin
             for kk:=1 to 5 do   write(a[k,kk]:5);
             writeln;
          end;
   end;
end;

procedure try(i,j,n:integer);          {以每一格为阶段，在每一阶段中试遍 8 个方向}
var
  k,x,y:integer;                       {这三个变量一定要定义局部变量}
begin
  if n>25 then begin print; exit;end ; {达到最大规模打印、统计方案}
  for k:=1 to 8 do                     {试遍 8 个方向}
   begin
     x:=i+u[k]; y:=j+v[k] ;            {走此方向，得到的新坐标}
     if (x<=5) and (x>=1) and (y<=5) and (y>=1)and b[x,y] then      {如果新坐标在
棋盘上，并且这一格可以走}
       begin
            b[x,y]:=false;
            a[x,y]:=n;
            try(x,y,n+1);                {从(x,y)去搜下一步该如何走}
            b[x,y]:=true;
            a[x,y]:=0;
       end;
   end;
end;

BEGIN
   u[1]:=1;     v[1]:=-2;               {8 个方向的 x,y 增量}
   u[2]:=2;     v[2]:=-1;
   u[3]:=2;     v[3]:=1;
   u[4]:=1;     v[4]:=2;
   u[5]:=-1;    v[5]:=2;
   u[6]:=-2;    v[6]:=1;
   u[7]:=-2;    v[7]:=-1;
   u[8]:=-1;    v[8]:=-2;
   for i:=1 to 5 do                {初始化}
     for j:=1 to 5 do
       begin
          a[i,j]:=0;
          b[i,j]:=true;
       end;
   a[1,1]:=1; b[1,1]:=false;        {从(1,1)第一步开始走}
```

```
   try(1,1,2);        {从(1,1)开始搜第 2 步该怎样走}
   writeln(num);      {输出总方案(304)}
END.
```

**【例 4】**六个城市之间道路联系的示意图如下图所示。连线表示两城市间有道路相通，连线旁的数字表示路程。请编写程序，有计算机找出从 $C_1$ 城到 $C_6$ 城的没有重复城市的所有不同路径，按照路程总长度的大小从小到大地打印出来这些路径。

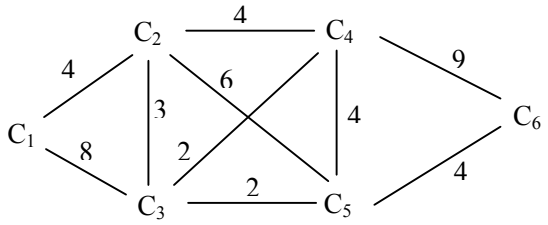
输出格式：

```
1: 1-->2-->5-->6          const=14
2: 1-->2-->3-->5-->6      const=15
3: 1-->3-->5-->6          const=16
……————————————————————————————..
```

**【算法分析】**

道路之间的联系可以用一个 6×6 的“邻接距阵”(即二维数组)LINK 来表示，LINK (i, j) 的值表示 $C_i$ 到 $C_j$ 城之间的路程，如果值等于零表示没有通路。

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0 | 4 | 8 | 0 | 0 | 0 |
| 2 | 4 | 0 | 3 | 4 | 6 | 0 |
| 3 | 8 | 3 | 0 | 2 | 2 | 0 |
| 4 | 0 | 4 | 2 | 0 | 4 | 9 |
| 5 | 0 | 6 | 2 | 4 | 0 | 4 |
| 6 | 0 | 0 | 0 | 9 | 4 | 0 |




建立产生式系统：其中数据库用数组 OPEN 做索引表，用 NODE（字符传数组）记录路径，LONG 记录路径长度；

产生式规则：R 为下一个城市编号，2≤R≤6，K 是当前路径，则有 5 条规则：

IF LINK（K[LENGTH（K）]，R）>0 且 $C_R$ 没有到过 THEN

增加一个 NODE 元素，把新增元素赋值为：K+R 。

增加一个 OPEN 元素，记下 NODE 元素的位置。

**【参考程序】**

```
program exam4;
const
   max=maxint;
   link:array[1..5,1..6] of integer=((0,4,8,0,0,0),(4,0,3,4,6,0),
                          (8,3,0,2,2,0),(0,4,2,0,4,9),(0,6,2,4,0,4));
                          {邻接表：最后一行可以省略，因为到达 C6 后不能再到别的城市}
type  path=string[6];  {字符串记录路径}
var
   open:array[1..6] of integer;  {索引表}
   node:array[1..100] of path;  {记录所有路径}
   count,i,n:integer;
```

```
procedure try(k,dep:integer);   {搜索过程}
var  r,len:byte;
        temp:path;
begin
  temp:=node[open[dep]];   {取出 NODE 表中最后一个元素}
  len:=length(temp);
  if pos('6',temp)>0 then exit    {不能再到别的城市}
  else
    for r:=2 to 6 do
       if(link[k,r]>0) and (pos(chr(48+r),temp)=0) then
         begin
             inc(n);node[n]:=temp+chr(48+r);
             open[dep+1]:=n;try(r,dep+1)    {搜索下一个城市}
         end
end;

procedure print;   {打印}
var
  f,i,j,k,l:integer;
  bool:array[1..100] of boolean;   {记录某路径是否已经打印}
  long:array[1..100] of integer;    {记录某路径的总长度}
begin
  count:=0;
  for i:=1 to n do
     if node[i,length(node[i])]<>'6' then
        bool[i]:=false
     else
          begin
            bool[i]:=true; inc(count);long[i]:=0;
            for j:=2 to length(node[i]) do
               long[i]:=long[i]+link[ord(node[i,j-1])-48,ord(node[i,j])-48];
               {统计长度}
          end;
          for i:=1 to count do
            begin
              k:=maxint;
              for j:=1 to n do
                 if (bool[j]) and (long[j]<k)then
                    begin k:=long[j];l:=j end;
              bool[l]:=false; write(i:2,':1');
              for j:=2 to length(node[l]) do write('-->',node[l,j]);  {输出路径}
              writeln('cost=',k)   {输出总长度}
            end;    {输出}
```

```
          readln
end;

begin
  n:=1;node[1]:='1';open[1]:=1;    {赋初始值}
  try(1,1);    {搜索}
  print;      {打印}
end.
```

非正常跳出语句：

HALT　　最高级，碰到整个程序就结束了。
EXIT　　跳出当前模块，回到调用它的语句后面。
BREAK　　在循环语句里面遇到 BREAK 则跳出当前的这一重循环。
CONTINUE　放弃这一次循环，继续下一次循环。

## 【上机练习】

### 1.植树

提交文件名：TREE.PAS

**问题描述：**

在 6*6 的方格地盘中，种植 24 颗树，使每行、每列都有 4 颗树。

**问题求解：**

求出所有可能的种植方案和方案总数？

### 2.字符序列

提交文件名：CHARACTS.PAS

**问题描述：**

从三个元素的集合[A，B，C]中选取元素生成一个 N 个字符组成的序列,使得没有两个相邻字的子序列相同。例：N = 5 时 ABCBA 是合格的，而序列 ABCBC 与 ABABC 是不合格的，因为其中子序列 BC，AB 是相同的。

**问题求解：**

对于由键盘输入的 N，求出满足条件的 N 个字符的所有序列和其总数？

### 3. 最小拉丁方阵

提交文件名：LATIN.PAS

**问题描述：**

输入 N，求 N 阶最小的拉丁方阵 (2 ≤ N ≤ 9)。N 阶拉丁方阵为每一行、每一列都是数字 1 到 N，且每个数字只出现一次。最小拉丁方阵是将方阵的一行一行数连接在一起，组成为一个数，则这个数是最小的。

输入输出示例：

```
N = 3
1 2 3
2 3 1
3 1 2
```

```
N = 5
1 2 3 4 5
2 1 4 5 3
3 4 5 1 2
4 5 2 3 1
5 3 1 2 4
```

## 4. 试卷批分 GRADE.PAS）

**问题描述：**

某学校进行了一次英语考试，共有 10 道是非题，每题为 10 分，解答用 1 表示“是”，用 0 表示“非”的方式。但老师批完卷后，发现漏批了一张试卷，而且标准答案也丢失了，手头只剩下了 3 张标有分数的试卷。

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 试卷一： | ① | ② | ③ | ④ | ⑤ | ⑥ | ⑦ | ⑧ | ⑨ | ⑩ | |
| | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 得分：70 |
| 试卷二： | ① | ② | ③ | ④ | ⑤ | ⑥ | ⑦ | ⑧ | ⑨ | ⑩ | |
| | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 得分：50 |
| 试郑三： | ① | ② | ③ | ④ | ⑤ | ⑥ | ⑦ | ⑧ | ⑨ | ⑩ | |
| | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 得分：30 |
| 待批试卷： | ① | ② | ③ | ④ | ⑤ | ⑥ | ⑦ | ⑧ | ⑨ | ⑩ | |
| | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 得分：？ |

**问题求解：**

请编一程序依据这三张试卷，算出漏批的那张试卷的分数

## 5. 皇后问题（QUEEN）

提交文件名：QUEEN.PAS

**问题描述：**

在 8＊8 的国际象棋的棋盘上，放置最少数目的皇后，使得这些皇后相互之间不能攻击，并能控制整个棋盘。

**问题求解：**

打印输出每一种放置方法和方法总数。

其中空格用‘-’表示，皇后用‘*’表示。

**【样例：】**

```
Queen=5    {皇后个数}
Total=728  {方法总数}
No.1:      {第一种放置方法}
 * - - - - - - -
 - - * - - - - -
 - - - - * - - -
 - * - - - - - -
 - - - * - - - -
 - - - - - - - -
 - - - - - - - -
 - - - - - - - -
```

## 6. 标尺问题（STAFF GUAGE）

提交文件名：STAFF.PAS

**问题描述：**

在一把 N 寸长的尺上（N ≤ 30），标记最少的刻度，使其能一次丈量 1 到 N 的所有整数长度。

**问题求解：**

对于由键盘输入的 N，求出满足条件的不同本质的解，并打印输出每一种标记法。

**【样例：】**

```
L = 10
No.  1 : Mark Place    1  2  3  6
No.  2 : Mark Place    1  2  3  7
No.  3 : Mark Place    1  2  4  7
No.  4 : Mark Place    1  2  4  9
No.  5 : Mark Place    1  2  5  7
No.  6 : Mark Place    1  2  5  8
No.  7 : Mark Place    1  2  6  7
No.  8 : Mark Place    1  2  6  9
No.  9 : Mark Place    1  3  4  8
No. 10 : Mark Place    1  3  4  9
No. 11 : Mark Place    1  3  5  9
No. 12 : Mark Place    1  3  6  8
No. 13 : Mark Place    1  3  6  9
No. 14 : Mark Place    1  3  7  8
No. 15 : Mark Place    1  4  5  8
No. 16 : Mark Place    1  4  6  8
No. 17 : Mark Place    1  5  6  8
No. 18 : Mark Place    1  5  7  8
No. 19 : Mark Place    1  6  7  8
T = 4
```

## 7. 棋盘问题（CHESS）

提交文件名：CHESS.PAS

**问题描述：**

在 5*5 的方格棋盘中，若在某一个方格内放入一个黑棋子，则与该方格相邻的上、下、左、右四个方格内不能再放白棋子。

**问题求解：**

请你设计一个程序，寻找并打印出所有放置 7 个黑棋子后，再也不能放一个白棋子的方案和方案总数。

**输出样例：**

```
No.1:                    No.2:
  *  -  *  -  -            *  -  -  *  -
  -  -  -  -  *            -  -  *  -  -
  -  *  -  -  -            -  -  -  -  *
  -  -  -  *  -            *  *  -  -  -
  *  -  -  *  -            -  -  -  *  -
Total=22
```

# 第十三章　广度优先搜索算法

## 一．宽度优先搜索的过程

宽度优先搜索算法（又称广度优先搜索）是最简便的图的搜索算法之一，这一算法也是很多重要的图的算法的原型。Dijkstra 单源最短路径算法和 Prim 最小生成树算法都采用了和宽度优先搜索类似的思想。

宽度优先算法的核心思想是：从初始节点开始，应用算符生成第一层节点，检查目标节点是否在这些后继节点中，若没有，再用产生式规则将所有第一层的节点逐一扩展，得到第二层节点，并逐一检查第二层节点中是否包含目标节点。若没有，再用算符逐一扩展第二层的所有节点……，如此依次扩展，检查下去，直到发现目标节点为止。即

1.从图中的某一顶点 V0 开始，先访问 V0；

2.访问所有与 V0 相邻接的顶点 V1，V2，.....，Vt；

3.依次访问与 V1，V2，.....，Vt 相邻接的所有未曾访问过的顶点；

4.循此以往，直至所有的顶点都被访问过为止。

这种搜索的次序体现沿层次向横向扩长的趋势，所以称之为广度优先搜索。

## 二、广度优先搜索算法描述：

```
Program Bfs;
初始化，初始状态存入 OPEN 表;
队列首指针 head:=0;尾指针 tail:=1;
repeat
    指针 head 后移一位，指向待扩展结点;
    for I=1 to max do      {max 为产生子结点的规则数}
    begin
      if 子结点符合条件 then
         begin
           tail 指针增 1，把新结点存入列尾;
           if 新结点与原已产生结点重复 then 删去该结点（取消入队，tail 减 1）
              else
                   if 新结点是目标结点 then 输出并退出;
            end;
        end;
until(head>=tail);    {队列空}
```

## 三、广度优先搜索注意事项：

1、每生成一个子结点，就要提供指向它们父亲结点的指针。当解出现时候，通过逆向跟踪，找到从根结点到目标结点的一条路径。

2、生成的结点要与前面所有已经产生结点比较，以免出现重复结点，浪费时间，还有可能陷入死循环。

3、如果目标结点的深度与“费用”（如：路径长度）成正比，那么，找到的第一个解即为最优解，这时，搜索速度比深度搜索要快些；如果结点的“费用”不与深度成正比时，第一次找到的解不一定是最优解。

4、广度优先搜索的效率还有赖于目标结点所在位置情况，如果目标结点深度处于较深层时，需搜索的结点数基本上以指数增长。

下面我们看看怎样用宽度优先搜索来解决八数码问题。

**例如**　图 2 给出广度优先搜索应用于八数码难题时所生成的搜索树。搜索树上的所有结点都标记它们所对应的状态，每个结点旁边的数字表示结点扩展的顺序。粗线条路径表明求得的一个解。从图中可以看出，扩展 2 6 个结点和生成 4 6 个结点之后，才求得这个解。此外，直接观察此图表明，不存在有更短走步序列的解。

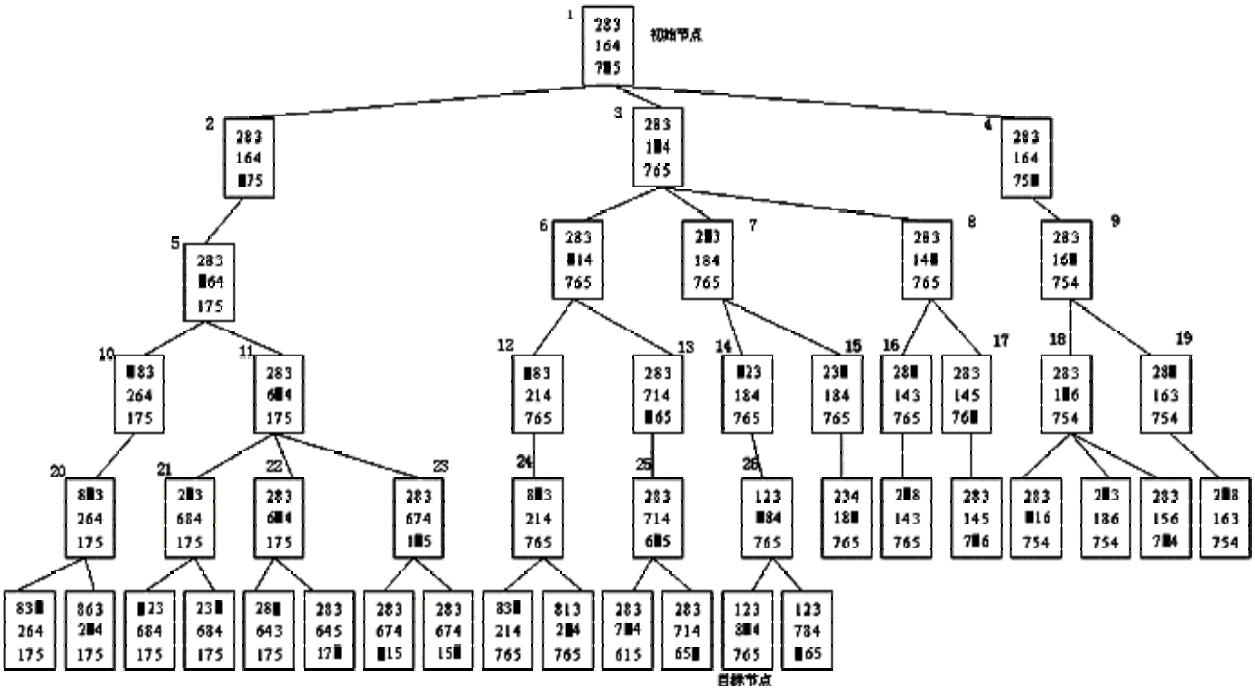


图 2　广度优先搜索图

**例 1：**图 4 表示的是从城市 A 到城市 H 的交通图。从图中可以看出，从城市 A 到城市 H 要经过若干个城市。现要找出一条经过城市最少的一条路线。

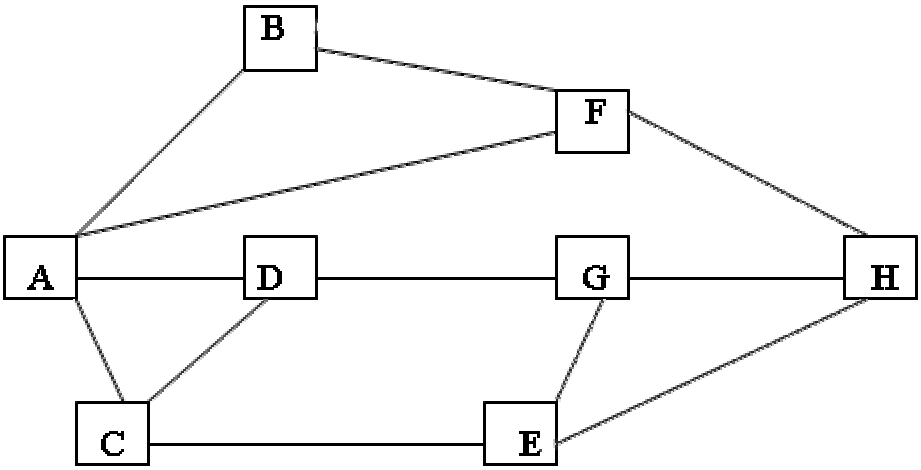



图 4

**【算法分析】**

看到这图很容易想到用邻接距阵来表示，0 表示能走，1 表示不能走。如图 5。

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| A | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| B | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| C | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| D | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| E | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| F | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| G | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| H | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |

图 5

首先想到的是用队的思想。我们可以 a 记录搜索过程，a.city 记录经过的城市,a.pre 记录前趋元素，这样就可以倒推出最短线路。具体过程如下：

（1） 将城市 A 入队，队首、队尾都为 1。

（2）将队首所指的城市所有可直通的城市入队（如果这个城市在队中出现过就不入队，可用一个集合来判断），将入队城市的 pre 指向队首的位置。然后将队首加 1，得到新的队首城市。重复以上步骤，直到城市 H 入队为止。当搜到城市 H 时，搜索结束。利用 pre 可倒推出最少城市线路。

**【参考程序】**

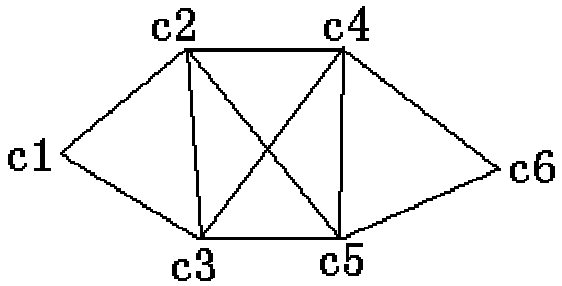
```
      const ju:array[1..8,1..8] of 0..1=((1,0,0,0,1,0,1,1),
                                         (0,1,1,1,1,0,1,1),
                                         (0,1,1,0,0,1,1,1),
                                         (0,1,0,1,1,1,0,1),
                                         (1,1,0,1,1,1,0,0),
                                         (0,0,1,1,1,1,1,0),
                                         (1,1,1,0,0,1,1,0),
                                         (1,1,1,1,0,0,0,1));
   type r=record {记录定义}
            city:array[1..100] of char;
             pre:array[1..100] of integer;
         end;
   var h,d,i:integer;
       a:r;
       s:set of 'A'..'H';

 procedure out; {输出过程}
   begin
     write(a.city[d]);
       repeat
         d:=a.pre[d];
         write('--',a.city[d]);
       until a.pre[d]=0;
```

```
      writeln;
      halt;
    end;
  procedure doit;
    begin
      h:=0; d:=1;
      a.city[1]:='A';
      a.pre[1]:=0;
      s:=['A'];
      repeat {步骤2}
        inc(h); {队首加一，出队}
        for i:=1 to 8 do {搜索可直通的城市}
          if （ju[ord(a.city[h])-64,i]=0）and
             （not（chr（i+64） in s））then {判断城市是否走过}
              begin
                inc(d); {队尾加一，入队}
                a.city[d]:=chr(64+i);
                a.pre[d]:=h;
                s:=s+[a.city[d]];
                if a.city[d]='H' then out;
              end;
      until h=d;
    end;
    BEGIN {主程序}
      doit;
    END.
```

**输出：**

```
      H--F--A
```

**例 2：**下面是六个城市之间道路联系的示意图，连线表示两城市之间有道路相通。请编一程序，由计算机找出从 C1 城到 C6 城的没有重复城市的所有不同的路径。




**【参考程序】**

```
const  link : array[1..5,1..6] of byte = ((0,1,1,0,0,0),
                                          (1,0,1,1,1,0),
                                          (1,1,0,1,1,0),
                                          (0,1,1,0,1,1),
```

```
                                        (0,1,1,1,0,1));
type  ft = set of 1..6;
var  qm{队列},pnt{parent 父节点} : array[1..100] of byte;
     f{走到这里通过的城市}      : array[1..100] of ft;
     fs     : ft;
     i,k,closed,open : byte;
procedure print;
  var
    n,i,j : byte;
    s : array[1..6] of byte;
begin
  i := open;  n := 0;
  while i > 0 do begin
    n := n+1;
    s[n] := i;
    i := pnt[i];
  end;
  write('Dep = ',n-1,' : 1');
  for j := n-1 downto 1 do begin
    write(' -> ',qm[s[j]]);
  end;
  readln;
end;
BEGIN
  f[1] := [1]   {城市 1 已到};
  qm[1] := 1    {队列的第一个节点是 1};
  pnt[1] := 0;   {没有父辈节点}
  closed := 0; open := 1;
  repeat
    inc(closed);
    k := qm[closed];
    if k <> 6 then begin
      fs := f[closed];
      for i := 2 to 6 do
        if (not(i in fs)) and (link[k,i] > 0) then begin
          inc(open);
          qm[open] := i;
          f[open] := fs+[i];
          pnt[open] := closed;
          if i = 6 then print;
        end;
    end;
  until closed >= open;
```

END.

在广度优先搜索中，我们将扩展出来的结点存贮在一个称作 qm 的数组里，qm 数组采用“先进先出”的队列结构，设两个指针 closed 和 open，分别是队首指针和队尾指针。其中 qm[1..closed-1]存贮已扩展的结点（即这些结点的子结点已扩展出）；qm[closed..open]存贮待扩展结点（即这些结点的子结点尚待扩展）。当 closed >= open 则表示队列空，结束。pnt 为父辈结点数组，它记录了每个结点的父辈结点，当找到目标后，可沿着父辈结点倒串上去，输出路径方案。在广度优先搜索中，第一个达到目标结点的，即是最短路径。

**例 3、最短路径（1995 年高中组第 4 题）**

如下图 5 所示，从入口（1）到出口（17）的可行路线图中，数字标号表示关卡。

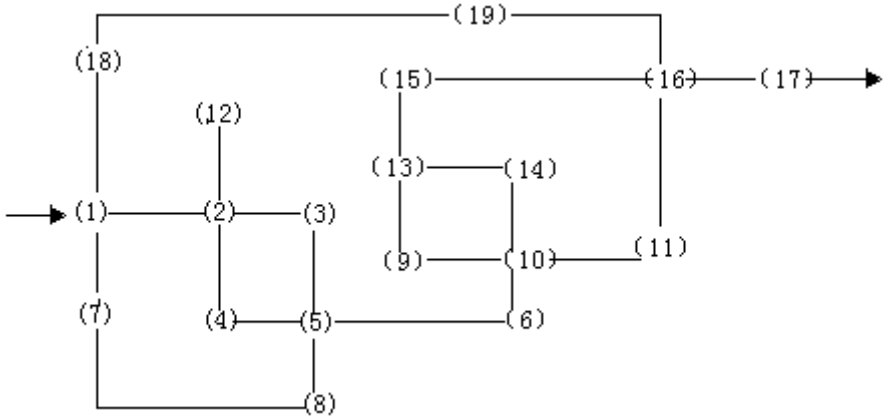



图 5

现将上面的路线图，按记录结构存储如下图 6：

| I | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | … | Max |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NO | 1 | 2 | 18 | 7 | 3 | 12 | 4 | 19 | 8 | 5 | 13 | 16 | 6 | 14 | 15 | 9 | 17 | | |
| PRE | 0 | 1 | 1 | 1 | 2 | 2 | 2 | 3 | 4 | 5 | 6 | 8 | 10 | 11 | 11 | 11 | 12 | | |

图 6

请设计一种能从存储数据中求出从入口到出口经过最少关卡路径的算法。

**【算法分析】**

该题是一个路径搜索问题，根据图示，从入口（1）到出口（17）可能有多条途径，其中最短的路径只有一条，那么如何找最短路径呢？根据题意，用数组 NO 存储各关卡号，用数组 PRE 存储访问到某关卡号的前趋关卡号。其实本题是一个典型的图的遍历问题，我们可以采用图的广度优先遍历，并利用队列的方式存储顶点之间的联系。从入口（1）开始先把它入队，然后把（1）的所有关联顶点都入队，即访问一个顶点，将其后继顶点入队，并存储它的前趋顶点，……，直到访问到出口（17）。最后，再从出口的关卡号（17）开始回访它的前趋关卡号，……，直到入口的关卡号（1），则回访的搜索路径便是最短路径。从列表中可以看出出口关卡号（17）的被访问路径最短的是：

（17）← （16）←（19）←（18）←（1）

由此，我们得到广度优先遍历求最短路径的基本方法如下：

假设用邻接矩阵存放路线图(a[I,j]=1 表示 I 与 j 连通,a[I,j]=0 表示 I 与 j 不连通)。再设一个队列和一个表示拓展到哪个顶点的指针变量 pos。

（1）从入口开始，先把（1）入队，并且根据邻接矩阵，把（1）的后继顶点全部入队，并存储这些后继顶点的前趋顶点为（1）；再把 pos 后移一个，继续拓展它，将其后继顶点入队，并存储它们的前趋顶点，……，直到拓展到出口（目的地（17））；

注意后继顶点入队前，必须要检查这个顶点是否已在队列中，如果已经在了就不要入队了；这一步可称为图的遍历或拓展，目的为形成队列；

（2）从队列的最后一个关卡号（出口（17））开始，依次回访它的前驱顶点，倒推所得到的路径即为最短路径。主要是依据每个顶点的前趋顶点倒推得到的。实现如下：

```
I：=1 ;
WHILE  NO[I]<> 17  DO  I:=I+1  ;
REPEAT
   WRITE( ‘( ‘,NO[I],’ )’ );
   WRITE( ‘←’ );
   I:=PRE[I] ;
UNTIL  I=0;
```

**【参考程序】** 略。

**例 4、迷宫问题 1**

如下图 7 所示，给出一个 N*M 的迷宫图和一个入口、一个出口。

编一个程序，打印一条从迷宫入口到出口的路径。这里黑色方块的单元表示走不通（用 -1 表示），白色方块的单元表示可以走（用 0 表示）。 只能往上、下、左、右四个方向走。如果无路则输出“no way.”。

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 入口 → | 0 | -1 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | |
| | 0 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | -1 | |
| | -1 | 0 | 0 | 0 | 0 | 0 | -1 | -1 | -1 | |
| | 0 | 0 | -1 | -1 | 0 | 0 | 0 | 0 | 0 | → 出口 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | -1 | |

**图 7**

**【算法分析】**

只要输出一条路径即可，所以是一个经典的回溯算法问题。实现见参考程序。

**【参考程序】**

```
program maze1;
const maxn=50;
var t:array [1..maxn,1..maxn] of integer;
    f:boolean;
    n,m,i,j,desx,desy,soux,souy,step:integer;
procedure  move(x,y:integer);
  begin
    t[x,y]:=step;  {走一步，作标记，把步数记下来}
    step:=step+1;  {步数+1}
```

```
    if (x=desx) and (y=desy)  then f:=true
      else begin
             if (y<>m) and (t[x,y+1]=0) then  move (x,y+1);        {优先向右走}
             if not f and (x<>n)and(t[x+1,y]=0) then move(x+1,y);{不能再优先往下走}
             if not f and (y<>1)and(t[x,y-1]=0) then move(x,y-1); {往左拐}
             if not f and (x<>1)and(t[x-1,y]=0) then move(x-1,y); {往上拐}
           end;
    if not f then t[x,y]:=0;  {回溯，清除走过的标记}
end;
BEGIN {main}
  write('input maze number n,m:');
  readln(n,m);  {n 行 m 列的迷宫}
  writeln('input maze:');
  for i:=1  to  n  do     {读入迷宫，0 表示通，-1 表示不通}
    begin
       for j:=1  to  m  do  read(t[i,j]);
       readln;
    end;
  write('input the enter:');  readln(soux,souy);  {入口}
  write('input the exit:');   readln(desx,desy); {出口}
  step:=1;     {初始化}
  f:=false;    {f=false 表示无解；f=true 表示找到了一个解}
  move(soux,souy);
  if f then
    for i:=1  to n do
      begin
        for  j:=1  to  m  do  write(t[i,j]:4);
        writeln;
      end
  else  writeln ('no way.');
END.
```

| 输入 1： | 输出 1： |
|---|---|
| 8 5<br>-1 -1 -1 -1 -1<br>0 0 0 0 -1<br>-1 -1 -1 0 -1<br>-1 0 0 0 -1<br>-1 0 0 -1 -1<br>-1 0 0 0 -1<br>-1 -1 -1 0 -1<br>-1 0 0 0 -1<br>2 1<br>8 4 | -1 -1 -1 -1 -1<br>1 2 3 4 -1<br>-1 -1 -1 5 -1<br>-1 0 7 6 -1<br>-1 0 8 -1 -1<br>-1 0 9 10 -1<br>-1 -1 -1 11 -1<br>-1 0 0 12 -1 |

| 输入 2： | 输出 2： |
|---|---|
| 8 5<br>-1 -1 -1 -1 -1<br>0 0 0 0 -1<br>-1 -1 -1 0 -1<br>-1 0 0 0 -1<br>-1 0 0 -1 -1<br>-1 0 0 0 -1<br>-1 -1 -1 -1 -1<br>-1 0 0 0 -1<br>2 1<br>8 4 | no way. |

**迷宫问题 2**

图 8 所示的迷宫图，其中阴影部分表示不通，处于迷宫中的每个位置都可以有 8 个方向探索可行路径前进，假设入口设在最左上角，出口位置设在最右下角，编写一个程序，找出一条从入口到出口的**最短路径**。如果不唯一，只要输出任一个即可。

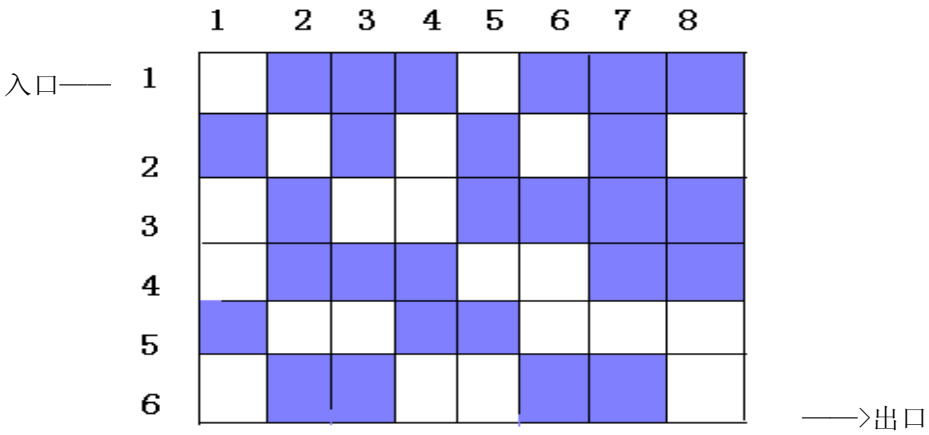



图 8

**【算法分析】**

（1）对于这样的迷宫依然可以用一个邻接矩阵表示（0 表示通，1 表示不通），如图 9 所示；为了克服在移动过程考虑出界的问题，我们给这个邻接矩阵增加了一圈，即人为地加上一个边界，如图 10 所示：

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |

图 9

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

总行数：0——m+1，　总列数：0——n+1

图 10

（2）从（X，Y）探索 8 个方向的表示如下：

| | | |
|---|---|---|
| ⑥ | ⑦ | ⑧ |
| ⑤ | X，Y | ① |
| ④ | ③ | ② |

8 个方向表示可以用一个增量数组：

```
Type    node= record
                x,y : -1..1 ;
            end;
var  F:array[1..8] of  node ;
```

| F | X | Y |
|---|---|---|
| 1 | 0 | 1 |
| 2 | 1 | 1 |
| 3 | 1 | 0 |
| 4 | 1 | -1 |
| 5 | 0 | -1 |
| 6 | -1 | -1 |
| 7 | -1 | 0 |
| 8 | -1 | 1 |

**图 11　增量数组**

（3）如何把探索的踪迹记录下来：用队列完成，与例 1 类似，其数据结构如下：

```
Type    sqtype=array[1..m*n]  of  record
                                      x,y : integer ;
                                      pre: 0..m*n  ;
                                  end;
var  sq : sqtype ;
```

（4）如何防止重复探索：将迷宫中的 0 替换为-1，队列中入队、出队情况如下表：

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| X | 1 | 2 | 3 | 3 | 3 | 2 | 4 | 4 | 1 | 5 | 4 | 5 | 2 | 5 | 6 | 5 | 6 | 6 | 5 | 6 |
| Y | 1 | 2 | 3 | 1 | 4 | 4 | 1 | 5 | 5 | 2 | 6 | 6 | 6 | 3 | 1 | 7 | 5 | 4 | 8 | 8 |
| PRE | 0 | 1 | 2 | 2 | 3 | 3 | 4 | 5 | 6 | 7 | 8 | 8 | 9 | 10 | 10 | 11 | 12 | 14 | 16 | 16 |
| 指针 | | | | | | | | | | | | | | | | F | | | | R |

（5）输出：（1,1）-->（2,2）-->（3,3）-->（3,4）-->（4,5）-->（4,6）-->（5,7）-->（6,8）

（6）**【参考程序】**略。

## 【上机练习】

**1、完成【例 1】--【例 4】的题目**

**2、硬币翻转(coin.pas)**

**【问题描述】**

在桌面上有一排硬币，共 N 枚，每一枚硬币均为正面朝上。现在要把所有的硬币翻转成反面朝上，规则是每次可翻转任意 N-1 枚硬币(正面向上的被翻转为反面向上，反之亦然)。求一个最短的操作序列（将每次翻转 N-1 枚硬币成为一次操作）。

**【输入格式】**

输入只有一行，包含一个自然数 N（N 为不大于 100 的*偶数*）。

**【输出格式】**

输出文件的第一行包含一个整数 S，表示最少需要的操作次数。接下来的 S 行每行分别表示每次操作后桌上硬币的状态（一行包含 N 个整数（0 或 1），表示每个硬币的状态：0——正面向上，和 1——反面向上，不允许出现多余空格）。

对于有多种操作方案的情况，则只需输出一种。

**【样例输入】**coin.in

4

**【样例输出】**coin.out

4

0111

1100

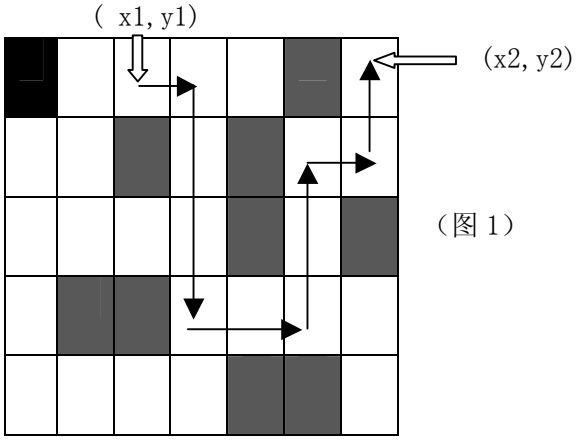
0001

1111

**6、最少转弯问题（TURN.PAS）**

**【问题描述】**

给出一张地图，这张地图被分为 n×m（n,m<=100）个方块，任何一个方块不是平地就是高山。平地可以通过，高山则不能。现在你处在地图的（x1,y1）这块平地，问：你至少需要拐几个弯才能到达目的地（x2,y2）？你只能沿着水平和垂直方向的平地上行进，拐弯次数就等于行进方向的改变（从水平到垂直或从垂直到水平）的次数。例如：如图 1，最少的拐弯次数为 5。




（图 1）

**【输入格式】**

第 1 行：n　m

第 2 至 n+1 行：整个地图地形描述（0：空地；1：高山），

如（图 1）第 2 行地形描述为：1 0 0 0 0 1 0

第 3 行地形描述为：0 0 1 0 1 0 0

……

第 n+2 行：x1　y1　x2　y2　（分别为起点、终点坐标）

**【输出格式】**

s （即最少的拐弯次数）

**【输入输出样例】（见图 1）：**

| TURN.IN | TURN.OUT |
|---|---|
| 5 7<br>1 0 0 0 0 1 0<br>0 0 1 0 1 0 0<br>0 0 0 0 1 0 1<br>0 1 1 0 0 0 0<br>0 0 0 0 1 1 0<br>1 3 1 7 | 5 |

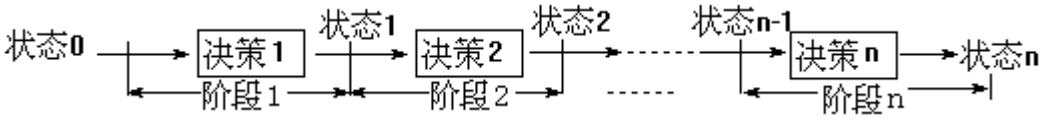
# 第十四章　动态规划

动态规划程序设计是对解最优化问题的一种途径、一种方法，而不是一种特殊算法。不象前面所述的那些搜索或数值计算那样，具有一个标准的数学表达式和明确清晰的解题方法。动态规划程序设计往往是针对一种最优化问题，由于各种问题的性质不同，确定最优解的条件也互不相同，因而动态规划的设计方法对不同的问题，有各具特色的解题方法，而不存在一种万能的动态规划算法，可以解决各类最优化问题。因此读者在学习时，除了要对基本概念和方法正确理解外，必须具体问题具体分析处理，以丰富的想象力去建立模型，用创造性的技巧去求解。我们也可以通过对若干有代表性的问题的动态规划算法进行分析、讨论，逐渐学会并掌握这一设计方法。

## 第一节 动态规划的基本模型

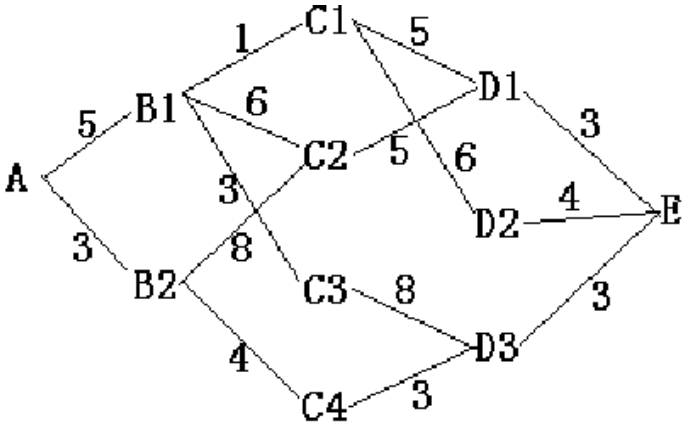
### 一、多阶段决策过程的最优化问题

在现实生活中，有一类活动的过程，由于它的特殊性，可将过程分成若干个互相联系的阶段，在它的每一阶段都需要作出决策，从而使整个过程达到最好的活动效果。当然，各个阶段决策的选取不是任何确定的，它依赖于当前面临的状态，又影响以后的发展，当各个阶段决策确定后，就组成一个决策序列，因而也就确定了整个过程的一条活动路线，这种把一个问题看作是一个前后关联具有链状结构的多阶段过程就称为多阶段决策过程，这种问题就称为多阶段决策问题。如下图所示：




多阶段决策过程，是指这样的一类特殊的活动过程，问题可以按时间顺序分解成若干相互联系的阶段，在每一个阶段都要做出决策，全部过程的决策是一个决策序列。要使整个活动的总体效果达到最优的问题，称为**多阶段决策问题**。

**例 1：最短路径问题。下图给出了一个地图，地图中的每个顶点代表一个城市，两个城市间的一条连线代表道路，连线上的数值代表道路的长度。现在想从城市 A 到达城市 E，怎样走路程最短？最短路程的长度是多少？**




**【算法分析】**把 A 到 E 的全过程分成四个阶段，用 K 表示阶段变量，第 1 阶段有一个初始状态 A，有两条可供选择的支路 A-B1、A-B2；第 2 阶段有两个初始状态 B1、B2，B1 有三条可

供选择的支路，B2 有两条可供选择的支路……。用 DK（$X_I$，$X+1_J$）表示在第 K 阶段由初始状态 $X_I$ 到下阶段的初始状态 $X+1_J$ 的路径距离，FK（$X_I$）表示从第 K 阶段的 $X_I$ 到终点 E 的最短距离，利用倒推的方法，求解 A 到 E 的最短距离。具体计算过程如下：

S1 ： K = 4 有

F4（D1）= 3，

F4（D2）= 4，

F4（D3）= 3；

S2 ： K = 3 有

F3（C1）= MIN{ D3（C1，D1）+ F4（D1），D3（C1，D2）+ F4（D2）}

= MIN{ 5+3，6+4 } = 8，

F3（C2）= D3（C2，D1）+ F4（D1）= 5+3 = 8；

F3（C3）= D3（C3，D3）+ F4（D3）= 8+3 = 11；

F3（C4）= D3（C4，D3）+ F4（D3）= 3+3 = 6；

S3 ： K = 2 有

F2（B1）= MIN{ D2（B1，C1）+ F3（C1），D2（B1，C2）+ F3（C2），

D2（B1，C3）+ F3(C3)} = MIN{ 1+8,6+8,3+11} = 9,

F2（B2）= MIN{ D2（B2，C2）+ F3（C2），D2（B2，C4）+ F3（C4）}

= MIN{ 8+8，4+6 } = 10；

S4 ： K = 1 有

F1（A）= MIN{ D1（A，B1）+ F2（B1），D1（A，B2）+ F2（B2）}

= MIN{ 5+9，3+10} = 13。

因此由 A 点到 E 点的全过程最短路径为 A→B2→C4→D3→E；最短路程长度为 13。

从以上过程可以看出，每个阶段中，都求出本阶段的各个初始状态到终点 E 的最短距离，当逆序倒推到过程起点 A 时，便得到了全过程的最短路径和最短距离。

在上例的多阶段决策问题中，各个阶段采取的决策，一般来说是与阶段有关的，决策依赖于当前状态，又随即引起状态的转移，一个决策序列就是在变化的状态中产生出来的，故有“动态”的含义，我们称这种解决多阶段决策最优化的过程为动态规划程序设计方法。

## 二、动态规划的基本概念和基本模型构成

现在我们来介绍动态规划的基本概念。

**1. 阶段和阶段变量：**

用动态规划求解一个问题时，需要将问题的全过程恰当地分成若干个相互联系的阶段，以便按一定的次序去求解。描述阶段的变量称为阶段变量，通常用 K 表示，阶段的划分一般是根据时间和空间的自然特征来划分，同时阶段的划分要便于把问题转化成多阶段决策过程，如例题 2 中，可将其划分成 4 个阶段，即 K = 1，2，3，4。

**2. 状态和状态变量：**

某一阶段的出发位置称为状态，通常一个阶段包含若干状态。一般地，状态可由变量来描述，用来描述状态的变量称为状态变量。如例题 2 中，C3 是一个状态变量。

**3. 决策、决策变量和决策允许集合：**

在对问题的处理中作出的每种选择性的行动就是决策。即从该阶段的每一个状态出发，通过一次选择性的行动转移至下一阶段的相应状态。一个实际问题可能要有多次决策和多个决策点，在每一个阶段的每一个状态中都需要有一次决策，决策也可以用变量

来描述，称这种变量为决策变量。在实际问题中，决策变量的取值往往限制在某一个范围之内，此范围称为允许决策集合。如例题 2 中，F3（C3）就是一个决策变量。

**4．策略和最优策略：**

所有阶段依次排列构成问题的全过程。全过程中各阶段决策变量所组成的有序总体称为策略。在实际问题中，从决策允许集合中找出最优效果的策略成为最优策略。

**5. 状态转移方程**

前一阶段的终点就是后一阶段的起点，对前一阶段的状态作出某种决策，产生后一阶段的状态，这种关系描述了由 k 阶段到 k+1 阶段状态的演变规律，称为状态转移方程。

## 三、最优化原理与无后效性

上面已经介绍了动态规划模型的基本组成，现在需要解决的问题是：什么样的“多阶段决策问题”才可以采用动态规划的方法求解。

一般来说，能够采用动态规划方法求解的问题，必须满足**最优化原理**和**无后效性原则**：

1、动态规划的最优化原理。作为整个过程的最优策略具有：无论过去的状态和决策如何，对前面的决策所形成的状态而言，余下的诸决策必须构成最优策略的性质。也可以通俗地理解为子问题的局部最优将导致整个问题的全局最优，即问题具有最优子结构的性质，也就是说一个问题的最优解只取决于其子问题的最优解，而非最优解对问题的求解没有影响。在例题 2 最短路径问题中，A 到 E 的最优路径上的任一点到终点 E 的路径，也必然是该点到终点 E 的一条最优路径，即整体优化可以分解为若干个局部优化。

2、动态规划的无后效性原则。所谓无后效性原则，指的是这样一种性质：某阶段的状态一旦确定，则此后过程的演变不再受此前各状态及决策的影响。也就是说，“未来与过去无关”，当前的状态是此前历史的一个完整的总结，此前的历史只能通过当前的状态去影响过程未来的演变。在例题 2 最短路径问题中，问题被划分成各个阶段之后，阶段 K 中的状态只能由阶段 K+1 中的状态通过状态转移方程得来，与其它状态没有关系，特别与未发生的状态没有关系，例如从 Ci 到 E 的最短路径，只与 Ci 的位置有关，它是由 Di 中的状态通过状态转移方程得来，与 E 状态，特别是 A 到 Ci 的路径选择无关，这就是无后效性。

由此可见，对于不能划分阶段的问题，不能运用动态规划来解；对于能划分阶段，但不符合最优化原理的，也不能用动态规划来解；既能划分阶段，又符合最优化原理的，但不具备无后效性原则，还是不能用动态规划来解；误用动态规划程序设计方法求解会导致错误的结果。

## 四、动态规划设计方法的一般模式

动态规划所处理的问题是一个多阶段决策问题，一般由初始状态开始，通过对中间阶段决策的选择，达到结束状态；或倒过来，从结束状态开始，通过对中间阶段决策的选择，达到初始状态。这些决策形成一个决策序列，同时确定了完成整个过程的一条活动路线，通常是求最优活动路线。

动态规划的设计都有着一定的模式，一般要经历以下几个步骤：

**1、划分阶段：**按照问题的时间或空间特征，把问题划分为若干个阶段。在划分阶段时，注意划分后的阶段一定是有序的或者是可排序的，否则问题就无法求解。

**2、确定状态和状态变量：**将问题发展到各个阶段时所处于的各种客观情况用不同的状态表示出来。当然，状态的选择要满足无后效性。

**3、确定决策并写出状态转移方程：**因为决策和状态转移有着天然的联系，状态转移就是根据上一阶段的状态和决策来导出本阶段的状态。所以如果确定了决策，状态转移方程也就可以写出。但事实上常常是反过来做，根据相邻两段的各个状态之间的关系来确定决策。

**4、寻找边界条件：**给出的状态转移方程是一个递推式，需要一个递推的终止条件或边界条件。

**5、程序的设计实现：**一旦设计完成，实现就会非常简单。下面我们给出从初始状态开始，通过对中间阶段决策的选择，达到结束状态，按照阶段、状态和决策的层次关系，写出的程序流程的一般形式：

```
所有状态费用的初始化；
for i := 阶段最大值-1 downto 1 do     {倒推每一个阶段}
  for j := 状态最小值 to 状态最大值 do  {枚举阶段 i 的每一个状态}
    for k := 决策最小值 to 决策最大值 do {枚举阶段 i 中状态 j 可选择的每一种决策}
      begin
        f[i,j] ← min{d[i,j,k] + f[i+1,k]}
      end;
输出 f[1,1];
```

**例 1** 对应的 Pascal 程序如下：

```
var  d : array[1..4,1..4,1..4] of byte;
     f : array[1..5,1..4] of byte;
     i,j,k,min : byte;
begin
  fillchar(d,sizeof(d),0);
  d[1,1,1] := 5; d[1,1,2] := 3;
  d[2,1,1] := 1; d[2,1,2] := 6; d[2,1,3] := 3;
  d[2,2,2] := 8; d[2,2,4] := 4;
  d[3,1,1] := 5; d[3,1,2] := 6;
  d[3,2,1] := 5;
  d[3,3,3] := 8;
  d[3,4,3] := 3;
  d[4,1,1] := 3;
  d[4,2,1] := 4;
  d[4,3,1] := 3;
  fillchar(f,sizeof(f),255);
  f[5,1] := 0;
  for i := 4 downto 1 do
    for j := 1 to 4 do
      for k := 1 to 4 do
        if d[i,j,k] <> 0 then
          if f[i,j] > d[i,j,k]+f[i+1,k] then f[i,j] := d[i,j,k]+f[i+1,k];
  writeln(f[1,1]);
  readln;
end.
```

# 第二节　动态规划与递推

## ——动态规划是最优化算法

由于动态规划的“名气”如此之大，以至于很多人甚至一些资料书上都往往把一种与动态规划十分相似的算法，当作是动态规划。这种算法就是递推。实际上，这两种算法还是很容易区分的。

按解题的目标来分，信息学试题主要分四类：判定性问题、构造性问题、计数问题和最优化问题。我们在竞赛中碰到的大多是最优化问题，而动态规划正是解决最优化问题的有力武器，因此动态规划在竞赛中的地位日益提高。而递推法在处理判定性问题和计数问题方面也是一把利器。下面分别就两个例子，谈一下递推法和动态规划在这两个方面的联系。

### 一、逆推法：

**例 2：数塔问题（IOI94）：**有形如图 1.3-8 所示的数塔，从顶部出发，在每一结点可以选择向左走或是向右走，一起走到底层，要求找出一条路径，使路径上的值最大。

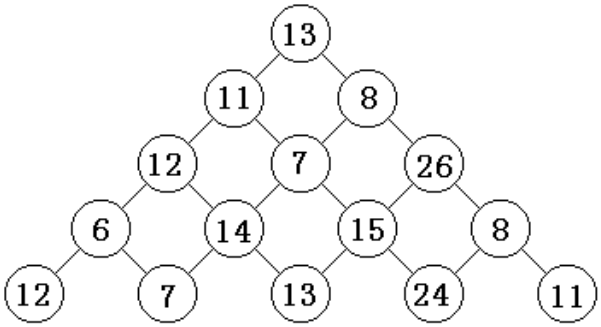



图 1.3-8

这道题如果用枚举法，在数塔层数稍大的情况下（如 40），则需要列举出的路径条数将是一个非常庞大的数目。如果用贪心法又往往得不到最优解。在用动态规划考虑数塔问题时可以自顶向下的分析，自底向上的计算。从顶点出发时到底向左走还是向右走应取决于是从左走能取到最大值还是从右走能取到最大值，只要左右两道路径上的最大值求出来了才能作出决策。同样的道理下一层的走向又要取决于再下一层上的最大值是否已经求出才能决策。这样一层一层推下去，直到倒数第二层时就非常明了。所以实际求解时，可从底层开始，层层递进，最后得到最大值。实际求解时应掌握其编程的一般规律，通常需要哪几个关键数组来存储变化过程这一点非常重要。

一般说来，很多最优化问题都有着对应的计数问题；反过来，很多计数问题也有着对应的最优化问题。因此，我们在遇到这两类问题时，不妨多联系、多发展，举一反三，从比较中更深入地理解动态规划的思想。

其实递推和动态规划这两种方法的思想本来就很相似，也不必说是谁借用了谁的思想。关键在于我们要掌握这种思想，这样我们无论在用动态规划法解最优化问题，或是在用递推法解判定型、计数问题时，都能得心应手、游刃有余了。

**【算法分析】**

①贪心法往往得不到最优解：本题若采用贪心法则：13-11-12-14-13，其和为 63

但存在另一条路：13-8-26-15-24，其和为 86。

贪心法问题所在：眼光短浅。

②动态规划求解：动态规划求解问题的过程归纳为：自顶向下的分析，自底向上计算。

**其基本方法是：**

划分阶段：按三角形的行，划分阶段，若有 n 行，则有 n-1 个阶段，找到问题求解的最优路径。

A. 从根结点 13 出发，选取它的两个方向中的一条支路，当到倒数第二层时，每个结点其后继仅有两个结点，可以直接比较，选择最大值为前进方向，从而求得从根结点开始到底端的最大路径。

B. 自底向上计算：（给出递推式和终止条件）

①从底层开始，本身数即为最大数；

②倒数第二层的计算，取决于底层的数据：12+6=18，13+14=27，24+15=39，24+8=32；

③倒数第三层的计算，取决于底二层计算的数据：27+12=39，39+7=46，39+26=65

④倒数第四层的计算，取决于底三层计算的数据：46+11=57，65+8=73

⑤最后的路径：13——8——26——15——24

C. 数据结构及算法设计

①图形转化：直角三角形，更于搜索：向下、向右

②用三维数组表示数塔：a[x,y,1]表示行、列及结点本身数据,a[x,y,2]能够取得最大值,a[x,y,3]表示前进的方向——0 向下，1 向右；

③算法：

数组初始化，输入每个结点值及初始的最大路径、前进方向为 0；

从倒数第二层开始向上一层求最大路径，共循环 N-1 次；

从顶向下，输出路径：关键是 J 的值，由于行逐渐递增，表示向下，究竟向下还是向右取决于列的值。若 J 值比原先多 1 则向右，否则向下。

**数塔问题的样例程序如下：**

```
var a:array[1..50,1..50,1..3] of longint;x,y,n:integer;
begin
    write( 'please input the number of rows:');
    readln(n);
    for x:=1 to n do
        for y:=1 to x do
          begin
            read(a[x,y,1]);
            a[x,y,2]:=a[x,y,1];
            a[x,y,3]:=0
          end;
    for x:=n-1 downto 1 do
      for y:=1 to x do
          if a[x+1,y,2]>a[x+1,y+1,2] then
              begin  a[x,y,2]:=a[x,y,2]+a[x+1,y,2]; a[x,y,3]:=0 end
          else  begin a[x,y,2]:=a[x,y,2]+a[x+1,y+1,2];a[x,y,3]:=1 end;
    writeln('max=',a[1,1,2]);
    y:=1;
    for x:=1 to n-1 do
      begin
        write(a[x,y,1],'-> ');
        y:=y+a[x,y,3]
```

```
    end;
  writeln(a[n,y,1])
end.
```

输入：

```
5     {数塔层数}
13
11    8
12    7     26
6    14     15     8
12    7     13    24     11
```

输出结果　max=86

13—8—26—15—24

## 例 3:求最长不下降序列

**㈠问题描述：**设有由 n 个不相同的整数组成的数列，记为:b(1)、b(2)、……、b(n)且 b(i)<>b(j)　(i<>j)，若存在 i1<i2<i3< … < ie 且有 b(i1)<b(i2)< … <b(ie)则称为长度为 e 的不下降序列。程序要求，当原数列给出之后，求出最长的不下降序列。

例如 13，7，9，16，38，24，37，18，44，19，21，22，63，15。例中 13，16，18，19，21，22，63 就是一个长度为 7 的不下降序列，同时也有 7 ，9，16，18，19，21，22，63 长度为 8 的不下降序列。

**㈡算法分析：**根据动态规划的原理，由后往前进行搜索。

1・对 b(n)来说，由于它是最后一个数，所以当从 b(n)开始查找时，只存在长度为 1 的不下降序列；

2・若从 b(n-1)开始查找，则存在下面的两种可能性：

①若 b(n-1)<b(n)则存在长度为 2 的不下降序列 b(n-1)，b(n)。

②若 b(n-1)>b(n)则存在长度为 1 的不下降序列 b(n-1)或 b(n)。

3・一般若从 b(i)开始，此时最长不下降序列应该按下列方法求出:

在 b(i+1),b(i+2),…,b(n)中，找出一个比 b(i)大的且最长的不下降序列，作为它的后继。

**㈢数据结构：**为算法上的需要，定义一个数组整数类型二维数组 b(N,3)

1・b(I,1)表示第 I 个数的数值本身；

2・b(I,2)表示从 I 位置到达 N 的最长不下降序列长度

3・b(I,3)表示从 I 位置开始最长不下降序列的下一个位置，若 b[I,3]=0 则表示后面没有连接项。

**㈣求解过程：**

①从倒数第二项开始计算，后面仅有 1 项，比较一次，因 63>15，不符合要求，长度仍为 1。

②从倒数第三项开始其后有 2 项，需做两次比较，得到目前最长的不下降序列为 2，如下表：

| | 11 | 12 | 13 | 14 | …… | | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|
| | | 22 | 63 | 15 | …… | | 21 | 22 | 63 | 15 |
| | | 2 | 1 | 1 | …… | | 3 | 2 | 1 | 1 |
| | | 13 | 0 | 0 | …… | | 12 | 13 | 0 | 0 |

**㈤一般处理过程是：**

①在 i+1,i+2,…,n 项中，找出比 b[I,1]大的最长长度 L 以及位置 K;

②若 L>0，则 b[I,2]:=L+1;b[I,3]:=k;

最后本题经过计算，其数据存储表如下：

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 13 | 7 | 9 | 16 | 38 | 24 | 37 | 18 | 44 | 19 | 21 | 22 | 63 | 15 |
| 7 | 8 | 7 | 6 | 3 | 4 | 3 | 5 | 2 | 4 | 3 | 2 | 1 | 1 |
| 4 | 3 | 4 | 8 | 9 | 7 | 9 | 10 | 13 | 11 | 12 | 13 | 0 | 0 |

**初始化:**

```
for i:=1 to n do
    begin
      read(b[i,1]);
      b[i,2]:=1;b[i,3]:=0;
    end;
```

**下面给出求最长不下降序列的算法:**

```
for i:=n-1 downto 1 do
      begin
         L:=0;k:=0;
         for j:=i+1 to n do
         if(b[j,1]>b[i,1])and(b[j,2]>L) then begin
                                                L:=b[j,2];k:=j;
                                              end;
         if L>0 then begin
                        b[i,2]:=L+1;b[i,3]:=k;
                     end;
      end;
```

**下面找出最长不下降序列：**

```
L:=1;
for j:=2 to n do
  if b[j,2]>b[L,2] then L:=j;
```

**最长不下降序列长度为 B(L, 2)序列**

```
while L<>0  do
  begin
    write(b[L,1]:4);
    L:=b[L,3];
  end;
```

**【参考程序】**

```
var
    n,i,L,k,j:integer;
    b:array[1..100,1..3]of integer;
begin
  writeln('input n:');
  readln(n);
  for i:=1 to n do
    begin
      read(b[i,1]);
      b[i,2]:=1;b[i,3]:=0;
    end;
  for i:=n-1 downto 1 do
      begin
         L:=0;k:=0;
         for j:=i+1 to n do
         if(b[j,1]>b[i,1])and(b[j,2]>L) then begin
                                                L:=b[j,2];
                                                k:=j;
                                              end;
         if L>0 then begin
                        b[i,2]:=L+1;b[i,3]:=k;
                     end;
      end;
      L:=1;
      for j:=2 to n do
         if b[j,2]>b[L,2] then L:=j;
      writeln('max=',b[L,2]);
        while L<>0  do
          begin
            write(b[L,1]:4);
            L:=b[L,3];
          end;
        writeln;
        readln;
end.
```

程序运行结果：

输入：13 7 9 16 38 24 37 18 44 19 21 22 63 15

输出：max=8

7  9  16  18  19  21  22  63

**例 4：拦截导弹**。某国为了防御敌国的导弹袭击，发展出一种导弹拦截系统。但是这种拦截系统有一个缺陷：虽然它的第一发炮弹能够到达任意的高度，但是以后每一发炮弹都不

能高于前一发的高度。某天，雷达捕捉到敌国的导弹来袭，由于该系统还在试用阶段。所以只有一套系统，因此有可能不能拦截所有的导弹。

输入导弹依次飞来的高度(雷达给出的高度不大于 30000 的正整数)。计算这套系统最多能拦截多少导弹。

输入：N 颗依次飞来的导弹高度，(导弹个数<=1000)。

输出：一套系统最多拦截的导弹数，并依次打印输出被拦截导弹的高度。

在本题中不仅要求输出最优解，而且还要求输出最优解的形成过程。为此，我们设置了一张记忆表 C[i]，在按从后往前方式求解的过程中，将每一个子问题的最佳决策保存起来，避免在输出方案时重复计算。

阶段 i：由右而左计算导弹 n‥导弹 1 中可拦截的最多导弹数（1≤i≤n）；

状态 B[i]：由于每个阶段中仅一个状态，因此可通过一重循环

for i := n-1 downto 1 do　枚举每个阶段的状态 B[i]；

决策 k：在拦截导弹 i 之后应拦截哪一枚导弹可使得 B[i]最大（i+1≤k≤n），

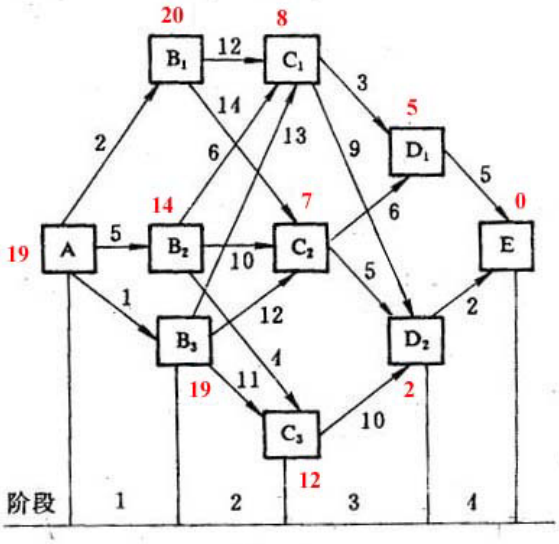
| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | I | |
| 13 | 7 | 9 | 16 | 38 | 24 | 37 | 18 | 44 | 19 | 21 | 22 | 63 | 15 | A[I] | {高度} |
| 2 | 1 | 1 | 2 | 4 | 3 | 3 | 2 | 3 | 2 | 2 | 2 | 2 | 1 | B[I] | {可拦截数} |
| 2 | 0 | 0 | 14 | 6 | 8 | 8 | 14 | 10 | 14 | 14 | 14 | 14 | 0 | C[I] | {再拦截} |

**【参考程序】**

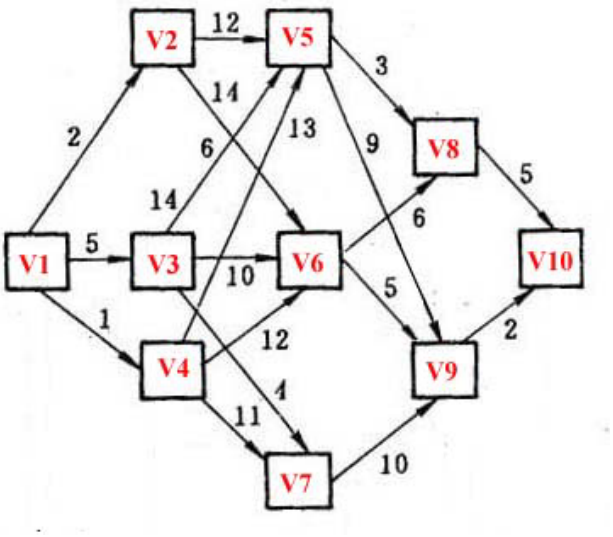
```
var
   a,b,c        : array[1..1000] of word;
   n,i,j,k,max : word;
begin
  n := 0;                          {初始化，读入数据}
  while not eoln do begin          {eoln : end of line}
    inc(n); read(a[n]); b[n] := 1; c[n] := 0;
  end;
  readln;
  for i := n-1 downto 1 do begin      {枚举每一个阶段的状态，设导弹 i 被拦截}
    max := 0; j := 0;
    for k := i+1 to n do              {枚举决策，计算最佳方案中拦截的下一枚导弹}
      if (a[k] <= a[i]) and (b[k] > max) then begin
        max := b[k]; j := k;
      end;
    b[i] := max+1; c[i] := j;         {若导弹 i 之后拦截导弹 j 为最佳方案，则记下}
  end;
  max := 0;
  for i := 1 to n do                  {枚举求出一套系统能拦截的最多导数}
    if b[i] > max then begin max := b[i]; j := i; end;
  writeln('OUTPUT');                  {打印输出结果}
  writeln('Max = ',b[j]);
  while j > 0 do begin
    write(a[j]:5); j := c[j];
  end;
  readln;
end.
```

例 5:下图表示城市之间的交通路网，线段上的数字表示费用，单向通行由 A->E。试用动态规划的最优化原理求出 A->E 的最省费用。

交通图 1




交通图 2




如图：求 v1 到 v10 的最短路径长度及最短路径。

**【样例输入】**

short.in

10

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 2 | 5 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 12 | 14 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 6 | 10 | 4 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 13 | 12 | 11 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 9 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 6 | 5 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**【样例输出】**

short.out

minlong=19

1 3 5 8 10

采用逆推法

设 f（x）表示点 x 到 v10 的最短路径长度

则 f(10)=0

f(x)=min{ f(i)+a[x,i] 当 a[x,i]>0 ,x<i<=n}

**【参考程序】**（逆推法）

```
program short;
var
 a:array[1..100,1..100] of integer;
 b,c:array[1..100] of integer;
 i,j,n,x:integer;
begin
 assign(input,'short.in');
 assign(output,'short.out');
 reset(input);rewrite(output);
 readln(n);
 for i:=1 to n do
   for j:=1 to n do
     read(a[i,j]);
 for i:=1 to n do
     b[i]:=maxint;
 b[n]:=0;
 for i:= n-1 downto 1 do
  for j:=n downto i+1 do
   if (a[i,j]>0) and (b[j]<>maxint) and (b[j]+a[i,j]<b[i])
    then begin b[i]:=b[j]+a[i,j];c[i]:=j end;
 writeln('minlong=',b[1]);
 x:=1;
  while x<>0 do
   begin
     write(x:5);
     x:=c[x];
   end;
  close(input);close(output);
end.
```

## 二、顺推法：

上面的例题都是逆推的，决策的选择也较多。下面我们来讲是顺推的，而且决策的选择较少。

**例 6：数塔问题。**设有一个三角形的数塔，如下图所示。顶点结点称为根结点，每个结点有一个整数数值，其值不超过 100。从顶点出发，可以向左走，也可以向右走。

```
13
11    8
12    7     26
6    14     15     8
12    7     13    24     11
```

从根 13 出发，向左走到达 11，再向右走到达 7，再向左走到达 14，再向左到达 7。由于 7 是最底层，无路可走。此时，我们找到一条从根结点开始到达底层的路径：13-11-7-14-7。路径上结点中数字的和，称为路径的值，如上面路径的值为 13+11+7+14+7 = 52。

当三角形数塔给出之后，找出一条路径，使路径上的值为最大，打印输出最大路径的值。数塔的层数 N 最多可为 100。

**【算法分析】**

**此题贪心法往往得不到最优解 ,例如 13-11-12-14-13 其路径的值为 63 ,但这不是最优解。**

穷举搜索往往是不可能的，当层数 N = 100 时，路径条数 P = $2^{99}$ 这是一个非常大的数，即使用世界上最快的电子计算机，也不能在短时间内计算出来。

对这道题唯一正确的方法是动态规划。如果得到一条由顶到底的某处的一条最佳路径，那么对于该路径上的每一个中间点来说，由顶至该中间点的路径所经过的数字和也为最大。因此本题是一个典型的多阶段决策最优化问题。在本题中仅要求输出最优解，为此我们设置了数组 A[i,j]保存三角形数塔，B[i,j]保存状态值，按从上往下方式进行求解。

阶段 i：以层数来划分阶段，由从上往下方式计算层数 1‥层数 N（1≤i≤n)；因此可通过第一重循环

for i := 1 to n do begin 枚举每一阶段；

状态 B[i,j]：由于每个阶段中有多个状态，因此可通过第二重循环

for j := 1 to i do begin 求出每个阶段的每个状态的最优解 B[i,j]；

决策：每个状态最多由上一层的两个结点连接过来，因此不需要做循环。

```
Var   a,b   : array[1..100,0..100] of word;
      i,j,n : byte;    max   : word;
begin
  repeat
    write('N = '); readln(n);
  until n in [1..100];
  fillchar(a,sizeof(a),0);
  b := a;
  for i := 1 to n do begin
    for j := 1 to i do read(a[i,j]);
    readln;
  end;
  b[1,1] := a[1,1];
  for i := 2 to n do
    for j := 1 to i do
      if b[i-1,j-1] > b[i-1,j]
        then b[i,j] := b[i-1,j-1]+a[i,j]
        else b[i,j] := b[i-1,j]+a[i,j];
  max := 0;
  for i := 1 to n do
    if b[n,i] > max then max := b[n,i];
  writeln('Max = ',max);
  readln;
end.
```

**例 7： 拦截导弹**

某国为了防御敌国的导弹袭击，发展出一种导弹拦截系统。但是这种导弹拦截系统有一个缺陷：虽然它的第一发炮弹能够到达任意的高度，但是以后每一发炮弹都不能高于前一发的高度。某天，雷达捕捉到敌国的导弹来袭。由于该系统还在试用阶段，所以只有一套系统，

因此有可能不能拦截所有的导弹。

输入导弹依次飞来的高度（雷达给出的高度数据是不大于 30000 的正整数），计算这套系统最多能拦截多少导弹，如果要拦截所有导弹最少要配备多少套这种导弹拦截系统。

样例：

| INPUT | OUTPUT |
|---|---|
| 389 207 155 300 299 170 158 65 | 6（最多能拦截的导弹数） |
| | 2（要拦截所有导弹最少要配备的系统数） |

**【算法分析】**

第一部分是要求输入数据串中的一个最长不上升序列的长度，可使用递推的方法，具体做法是从序列的第一个元素开始，依次求出以第 i 个元素为最后一个元素时的最长不上升序列的长度 s(i)，递推公式为 s(1)=1，s(i)=max(s(j)+1)，其中 i>1，j=1,2,…i-1，且 j 同时要满足条件：序列中第 j 个元素大于等于第 i 个元素。

第二部分比较有意思。由于它紧接着第一问，所以很容易受前面的影响，采取多次求最长不上升序列的办法，然后得出总次数，其实这是不对的。要举反例并不难，比如长为 7 的高度序列“7 5 4 1 6 3 2”， 最长不上升序列为“7 5 4 3 2”，用多次求最长不上升序列的结果为 3 套系统；但其实只要 2 套，分别击落“7 5 4 1”与“6 3 2”。那么，正确的做法又是什么呢？

我们的目标是用最少的系统击落所有导弹，至于系统之间怎么分配导弹数目则无关紧要；上面错误的想法正是承袭了“一套系统尽量多拦截导弹”的思维定势，忽视了最优解中各个系统拦截数较为平均的情况，本质上是一种贪心算法。如果从每套系统拦截的导弹方面来想行不通的话，我们就应该换一个思路，从拦截某个导弹所选的系统入手。

题目告诉我们，已有系统目前的瞄准高度必须不低于来犯导弹高度，所以，当已有的系统均无法拦截该导弹时，就不得不启用新系统。如果已有系统中有一个能拦截该导弹，我们是应该继续使用它，还是另起炉灶呢？事实是：无论用哪套系统，只要拦截了这枚导弹，那么系统的瞄准高度就等于导弹高度，这一点对旧的或新的系统都适用。而新系统能拦截的导弹高度最高，即新系统的性能优于任意一套已使用的系统。既然如此，我们当然应该选择已有的系统。如果已有系统中有多个可以拦截该导弹，究竟选哪一个呢？当前瞄准高度较高的系统的“潜力”较大，而瞄准高度较低的系统则不同，它能打下的导弹别的系统也能打下，它够不到的导弹却未必是别的系统所够不到的。所以，当有多个系统供选择时，要选瞄准高度最低的使用，当然瞄准高度同时也要大于等于来犯导弹高度。

解题时，用一个数组记下已有系统的当前瞄准高度，数据个数就是系统数目。

**解法一：**
```
program sheep506;
var
    i,n,j,best,num,x:integer;
    a,s,b:array[0..1000] of integer;
    f:text;
 begin
  assign(f,'cz2.txt');
  reset(f);
  readln(f,n);
  for i:=1 to n do
   begin
     read(f,a[i]);
      s[i]:=1;
   end;
```

```
close(f);
for i:=2 to n do
 begin
  for j:=1 to i-1 do
   if (a[j]>=a[i]) and (s[j]+1>s[i]) then
     s[i]:=s[j]+1;
   if s[i]>best then best:=s[i];
 end;
writeln('Max:',best);
num:=0;
b[0]:=9999;
for i:=1 to n do
 begin
  x:=0;
  for j:=1 to num do
   if (b[j]>a[i]) and (b[j]<b[x]) then x:=j;
  if x=0 then begin inc(num);b[num]:=a[i] end
  else b[x]:=a[i];
 end;
writeln(num);
readln;
end.
```

第一问经过计算，其数据存储表如下

| | A[1] | A[2] | A[3] | A[4] | A[5] | A[6] | A[7] | A[8] |
|---|---|---|---|---|---|---|---|---|
| | 389 | 207 | 155 | 300 | 299 | 170 | 158 | 65 |
| | S[1] | S[2] | S[3] | S[4] | S[5] | S[6] | S[7] | S[8] |
| I=1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| I=2 | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 1 |
| I=3 | 1 | 2 | 3 | 1 | 1 | 1 | 1 | 1 |
| I=4 | 1 | 2 | 3 | 2 | 1 | 1 | 1 | 1 |
| I=5 | 1 | 2 | 3 | 2 | 3 | 1 | 1 | 1 |
| I=6 | 1 | 2 | 3 | 2 | 3 | 4 | 1 | 1 |
| I=7 | 1 | 2 | 3 | 2 | 3 | 4 | 5 | 1 |
| I=8 | 1 | 2 | 3 | 2 | 3 | 4 | 5 | 6 |

第二问经过计算，其数据存储表如下：

| | num | b[0] | b[1] | b[2] | b[3] | b[4] |
|---|---|---|---|---|---|---|
| I=1 | 1 | 9999 | 389 | | | |
| I=2 | | | 207 | | | |
| I=3 | | | 155 | | | |
| I=4 | 2 | | | 300 | | |
| I=5 | | | | 299 | | |
| I=6 | | | | 170 | | |
| I=7 | | | | 158 | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| I=8 | | | 65 | | | |

**解法二：**

第一问即经典的最长不下降子序列问题，可以用一般的 DP 算法，也可以用高效算法，但这个题的数据规模好像不需要。

高效算法是这样的：用 a[x]表示原序列中第 x 个元素，b[x]表示长度为 x 的不下降子序列的最后一个元素的最小值，b 数组初值为无穷大。容易看出，这个数组是递减的（当然可能有相邻两个元素相等）。当处理第 a[x]时，用二分法查找它可以连接到长度最大为多少的不下降子序列后（即与部分 b[x]比较）。假设可以连到长度最大为 y 的不下降子序列后，则 b[y+1]:=min(b[y+1],a[x])。最后，b 数组被赋值的元素最大下标就是第一问的答案。由于利用了二分查找，这种算法的复杂度为 O(nlogn)，优于一般的 O(n^2)。

第二问用贪心法即可。每颗导弹来袭时，使用能拦截这颗导弹的防御系统中上一次拦截导弹高度最低的那一套来拦截。若不存在符合这一条件的系统，则使用一套新系统。

```
program tju1004;
const
  max=20;
var
  a,b,h:array[1..max]of word;
  i,j,m,n,x:byte;
begin
  repeat
    inc(i);
    read(a[i]);
    for j:=1 to i-1 do
      if a[j]>=a[i] then
        if b[j]>b[i] then b[i]:=b[j];
    inc(b[i]);
    if b[i]>m then m:=b[i];
    x:=0;
    for j:=1 to n do
      if h[j]>=a[i] then
        if x=0 then x:=j
            else if h[j]<h[x] then x:=j;
    if x=0 then begin inc(n);x:=n;end;
    h[x]:=a[i];
  until seekeof;
  writeln(m,' ',n);
end.
```

经过计算，其数据存储表如下

| I | I=1 | I=2 | I=3 | I=4 | I=5 | I=6 | I=7 | I=8 |
|---|---|---|---|---|---|---|---|---|
| A[I] | 389 | 207 | 155 | 300 | 299 | 170 | 158 | 65 |
| B[I] | 1 | 2 | 3 | 2 | 3 | 4 | 5 | 6 |
| N 值 | 1 | | | 2 | | | | |
| H[1] | 389 | 207 | 155 | | | | | 65 |
| H[2] | | | | 300 | 299 | 170 | 158 | |

# 第三节　动态规划经典例题

**【例 1】骑士游历问题**

设有一个 n*m 的棋盘(2≤n≤50，2≤m≤50)，如图 1 1.2.1。在棋盘上任一点有一个中国象棋马，

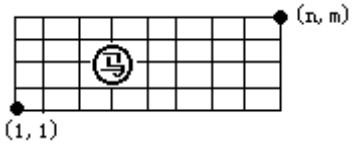



图 1 1.2.1

马走的规则为：

1.马走日字　2.马只能向右走。即图 1 1.2.2 所示：

图 1 1.2.2

当 N，M 给出之后，同时给出马起始的位置和终点的位置，试找出从起点到终点的所有路径的数目。例如：(N=10，M=10)，(1，5)(起点)，(3，5)(终点)。应输出 2(即由(1，5)到(3，5)共有 2 条路径，如图 1 1.2.3)：

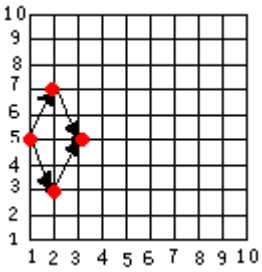



图 1 1.2.3

**输入：**

n，m，x1，y1，x2，y2(分别表示 n，m，起点坐标，终点坐标)

**输出：**

路径数目(若不存在从起点到终点的路径，输出 0)

**【算法分析】**

使用回溯法同样可以计算路径数目。只要将起点（1，1）和终点（n，m）调整为（x1，y1)、(x2，y2)，并在回溯程序(search(k，x，y))中，将马跳到目的地时由退出程序（halt）改为回溯（exit）即可。但问题是搜索效率太低，根本不可能在较短的时间内出解。本题并不要求每一条路径的具体走法。在这种情况下，是否非得通过枚举所有路径方案后才能得出路径数目，有没有一条简便和快效的“捷径”呢。

从($x_1$，$y_1$)出发，按照由左而右的顺序定义阶段的方向。位于（x，y）左方且可达（x，y）的跳马位置集合都是（x，y）的子问题，起点至（x，y）的路径数实际上等于起点至这些位置集的路径数之和（图 11.2.4）。

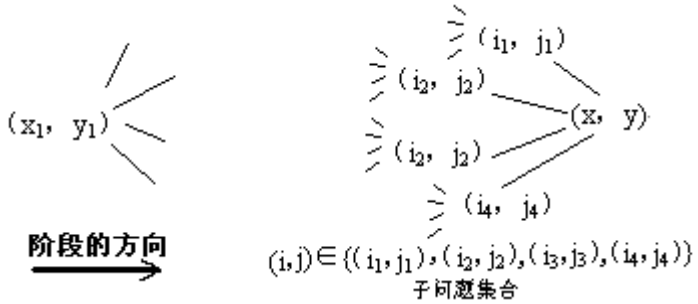



图 11.2.4

如此一来，状态转移关系便凸显出来。设状态转移方程 map，其中 map[i，j]为起点（x1，y1）至（i，j）的路径数目。由于棋盘规模的上限为 50*50，可能导致路径数目大得惊人，因此不妨设 map 数组的元素类型为 extended。初始时，除 map[x1，y1]=1 外其余为 0 。显然

$$map[x,y]=\sum_{(i,j)\in\text{可达}(x,y)\text{的坐标集}}\{map[i,j]+map[x,y]\,|\,(x,y)\text{在界内}\}\text{。}$$

我们采用动态程序设计的方法计算起点(x1，y1)至终点（x2，y2）的路径数目 map[x2，y2]：

阶段 j：中国象棋马当前的列位置(y1≤j≤y2)；

状态 i：中国象棋马在 j 列的行位置（1≤i≤n)；

决策 k：中国象棋马在（i，j）的起跳方向(1≤k≤4)；

计算过程如下：

```
fillchar(map，sizeof(map)，0)；
map[x1，y1] ←1；                                  {从（x1，y1）出发}
for j←y1 to y2 do                                 {递推中国象棋马的列位置}
  for i←1 to n do                                 {递推中国象棋马在 j 列的行位置}
    for k←1 to 4 do                               {递推中国象棋马在（i，j）的 4 个跳动方向}
      begin
        中国象棋马由（i，j）出发，沿着 k 方向跳至（x，y)；
        if (x∈ { 1..n} )∧(y∈{1..y2})              {计算状态转移方程}
          then map[x，y] ←map[i，j]+map[x，y]
      end；{for}
writeln(map[x2，y2]：0：0)；          {输出从（x1，y1）到（x2，y2）的路径数目}
```

## 【例 2】砝码称重

设有 1g，2g，3g，5g，10g，20g 的砝码各若干枚（其总重≤1000g)，要求：

**输入：**

a1　a2　a3　a4　a5　a6(表示 1g 砝码有 a1 个，2g 砝码有 a2 个，....20g 砝码有 a6 个)

**输出：**

Total=N (N 表示用这些砝码能称出的不同重量的个数，但不包括一个砝码也不用的情况)

**输入样例：**

1 1 0 0 0 0

**输出样例：**

Total=3，表示可以称出 1g，2g，3g 三种不同的重量

**【算法分析】**

设

```
const num：array[1..6] of shortint=(1，2，3，5，10，20)；        {砝码的重量序列}
var
  a：array[1..6] of integer；                                    { 6 种砝码的个数}
  visited：array[0..1000] of boolean；                          {重量的访问标志序列}
  n：array[0..1000] of integer；{n[0]—不同重量数；n[j]—第 j 种重量（1 ≤j≤n[0])}
  total，i，j，k：integer；                                     {total—目前称出的重量}
```

我们按照第 1 种砝码，第 2 种砝码……第 6 种砝码的顺序分析。在分析第 i 种砝码的放置方案时，依次在现有的不同重量的基础上，放 1 块、2 块……a[i]块，产生新的不同重量。n[n[0]+1]=total | total=n[j]+k*num[i]，visited[total]=false，1 ≤i≤6，1 ≤j≤n[0]， 1 ≤k≤a[i]

阶段 i： 分析第 i 种砝码(1 ≤i≤6)；

状态 j：枚举现有的不同重量（1 ≤j≤n[0])；

决策 k：在现有重量的基础上放 k 块第 i 种砝码，产生重量 n[j]+k*num[i](1 ≤k≤a[i])；

计算过程如下：

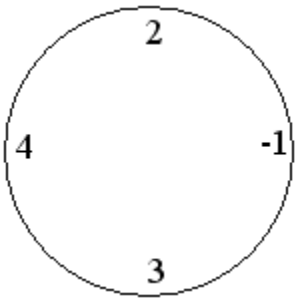
```
  fillchar(visited，sizeof(visited)，false)；
  write(’a1-a6:’)；for i←1 to 6 do read(a[i])；          {输入 6 种砝码的个数}
  n[0] ←1；n[1] ←0；                                      {产生第 1 种重量 0 }
  for i←1 to 6 do                                        {阶段：分析第 i 种砝码}
    for j←1 to n[0] do                                   {状态：枚举现有的不同重量}
      for k←1 to a[i] do          {决策：在现有重量的基础上放 k 块第 i 种砝码}
      begin
        total←n[j]+k*num[i]；                                        {产生重量}
        if not visited[total] then              {若该重量未产生过，则设访问标志}
        begin
          visited[total] ←true；
          inc(n[0])；                                          重量进入 n 序列}
          n[n[0]] ←total；
        end；{then}
      end；{for}
  writeln(n[0]-1)；                                      {输出不同重量的个数}
```

## 【例 3】数字游戏

**【问题描述】**

丁丁最近沉迷于一个数字游戏之中。这个游戏看似简单，但丁丁在研究了许多天之后却发觉原来在简单的规则下想要赢得这个游戏并不那么容易。游戏是这样的，在你面前有一圈

整数（一共 n 个），你要按顺序将其分为 m 个部分，各部分内的数字相加，相加所得的 m 个结果对 10 取模后再相乘，最终得到一个数 k。游戏的要求是使你所得的 k 最大或者最小。例如，对于下面这圈数字（n=4，m=2）：




当要求最小值时，((2-1) mod 10)×((4+3) mod 10)=1×7=7，要求最大值时，为((2+4+3) mod 10)×(-1 mod 10)=9×9=81。特别值得注意的是，无论是负数还是正数，对 10 取模的结果均为非负值。

丁丁请你编写程序帮他赢得这个游戏。

**【输入格式】**

输入文件第一行有两个整数，n（1≤n≤50）和 m（1≤m≤9）。以下 n 行每行有个整数，其绝对值不大于 $10^4$，按顺序给出圈中的数字，首尾相接。

**【输出格式】**

输出文件有两行，各包含一个非负整数。第一行是你程序得到的最小值，第二行是最大值。

**【输入样例】**

```
4 2
4
3
-1
2
```

**【输出样例】**

```
7
81
```

**【算法分析】**

设圆周上的 n 个数字为 $a_1$、$a_2$、…$a_n$。按照模运算的规则（$a_1+a_2+ \cdots +a_k$）mod 10=($a_1$ mod 10+$a_2$ mod 10+ $a_k$ mod 10)mod 10；g[i，j]为 $a_i$、$a_{i+1}$、…$a_j$ 的数和对 10 的模，即

g[i，j]=（$a_i+a_{i+1}+ \cdots +a_j$）mod 10

显然

g[i，i]=（$a_i$ mod 10+10）mod 10

g[i，j]= (g[i，j-1]+g[j，j]) mod 10　　(1≤i≤n，i+1≤j≤n)

当 m=1 时，程序得到的最小值和最大值为 g[1，n]；

fmax1[p，I，j]为 $a_i$、$a_{i+1}$、…$a_j$ 分为 p 个部分，各部分内的数字相加，相加所得的 p 个结果对 10 取模后再相乘，最终得到最大数。显然，fmax1[1，i，j]= g[i，j]；

fmin1[p，I，j]为 $a_i$、$a_{i+1}$、…$a_j$ 分为 p 个部分，各部分内的数字相加，相加所得的 p

个结果对 10 取模后再相乘，最终得到最小数。显然，fmin1[1，i，j]= g[i，j]；

(1≤p≤m)

问题是当 $a_i$、$a_{i+1}$、…$a_j$ 划分成的部分数 p 大于 1 时，怎么办。我们采用动态程序设计的方法计算。设

阶段 p：$a_i$、$a_{i+1}$、…$a_j$ 划分成的部分数，2≤p≤m-1；

状态（i，j）:将 $a_i$、$a_{i+1}$、…$a_j$ 划分成 p 个部分，1≤i≤n，i≤j≤n；

决策 k: 将 $a_i$、$a_{i+1}$、…$a_k$ 划分成 p-1 个部分，$a_{k+1}$、…$a_j$ 为第 p 部分，i≤k≤j-1；

显然，状态转移方程为

fmin1[p， i， j]= $\min\limits_{i\le k\le j-1}$ {fmin1[p-1， i， k]*g[k+1， j]}　　fmax1[p， i， j]= $\max\limits_{i\le k\le j-1}$ {fmax1[p-1，i，k]*g[k+1，j]}

2≤p≤m-1

按照上述公式递推出 $a_i$、$a_{i+1}$、…$a_j$ 划分成 m-1 个部分的最大值 fmax1[m-1，i，j]和最小值 fmin1[m-1，i，j]。由于圆周上的 n 个数首尾相接，因此第 m 部分设为 $a_1$…$a_{i-1}$、$a_{j+1}$…$a_n$。显然 $a_1$、$a_2$、…$a_n$ 划分成 m 个部分的最大值 max 和最小值 min 为

$$\max=\max_{\substack{1\le i\le n\\ i\le j\le n}}\{(g[1,i-1]+g[j+1,n)\bmod 10*f\max 1[m-1,i,j]|(i\ne 1)or(j\ne n)\}$$

$$\min=\min_{\substack{1\le i\le n\\ i\le j\le n}}\{(g[1,i-1]+g[j+1,n)\bmod 10*f\min 1[m-1,i,j]|(i\ne 1)or(j\ne n)\}$$

由于 p-1 阶段仅和 p 阶段发生联系，因此我们将 p-1 阶段的状态转移方程 fmin1[p-1，i，j]设为 fmin1[i,j]、fmax1[p-1,i,j] 设为 fmax1[i,j]，将 p 阶段的状态转移方程 fmin1[p，i，j]设为 fmin[i，j]、fmax1[p，i，j]设为 fmax[i，j]。由此得出算法：

```
read(n，m);{读数字个数和划分的部分数}
fillchar(fmax1，sizeof(fmax1)，0); fillchar(fmin1，sizeof(fmin1)，$FF);
                                                    {状态转移方程初始化}
fillchar(g，sizeof(g)，0);
for i:=1 to n do                          {依次读入每个数，一个数组成一部分}
    begin
     read(g[i，i]);
     g[i，i]:=(g[i，i] mod 10+10) mod 10;
     fmin1[i，i]:=g[i，i];fmax1[i，i]:=g[i，i];
    end;{for}
for i:=1 to n do                    {计算一部分内的数和对 10 的模的所有可能情况}
  for j:=i+1 to n do
   begin
    g[i，j]:=(g[i，j-1]+g[j，j]) mod 10;
    fmax1[i，j]:=g[i，j];fmin1[i，j]:=g[i，j];
   end;{for}
for p:=2 to m-1 do                {阶段：递推计算划分 2 部分…m-1 部分的结果值}
 begin
  fillchar(fmax,sizeof(fmax),0);            { 划分 p 部分的状态转移方程初始化}
```

```
  fillchar(fmin, sizeof(fmin), $FF);
  for i:=1 to n do                          {状态：枚举被划分为 p 部分的数字区间}
   for j:=i to n do
    for k:=i to j-1 do                      {决策：ai…ak被划分为 p-1 部分}
     begin
      if fmax1[i, k]*g[k+1, j]>fmax[i, j] then
                                {计算将 ai、ai+1、…aj 划分成 p 个部分的状态转移方程}
        fmax[i, j]:=fmax1[i, k]*g[k+1, j];
      if(fmin1[i,k]>=0)and((fmin1[i,k]*g[k+1,j]<fmin[i,j])or(fmin[i,j]<0))
          then fmin[i, j]:=fmin1[i, k]*g[k+1, j];
     end;{for}
   fmin1:=fmin;fmax1:=fmax;
end;{for}
max:=0;min:=maxlongint;  {将 a1、a2、…an 划分成 m 个部分的最大值和最小值初始化}
if m=1
  then                              {计算 n 个数划分成一部分的最大值和最小值}
      begin max:=g[1, n];min:=g[1, n];end{then}
  else for i:=1 to n do{将 a1…ai-1、aj+1…an   设为第 m 部分，计算最大值和最小值}
        for j:=i to n do
        if (i<>1) or (j<>n) then
        begin
         if (g[1, i-1]+g[j+1, n]) mod 10*fmax1[i, j]>max
            then max:=(g[1, i-1]+g[j+1, n]) mod 10*fmax1[i, j];
         if(fmin1[i, j]>=0) and ((g[1, i-1]+g[j+1, n]) mod 10*fmin1[i, j]<min)
            then min:=(g[1, i-1]+g[j+1, n]) mod 10*fmin1[i, j];
        end;{then}
writeln(min); writeln(max);                          {输出最小值和最大值}
```

本题的计算过程分两个阶段

第一个阶段：将圆周上的 n 个数排成一个序列，计算 $a_i$、$a_{i+1}$、…$a_j$ 划分成 m-1 个部分的最大值 fmax1[i, j]和最小值 fmin1[i, j]；

第二个阶段：将序列首尾相接。枚举第 m 部分的所有可能情况，在 fmax1 和 fmin1 的基础上，计算圆周上的 n 个数划分成 m 个部分的最大值 max 和最小值 min。

能否将两个阶段合并，用一个状态转移方程来解决呢？可以，这个问题留给读者思考。动态程序设计方法是多样性的。我们在用动态程序设计方法解题的时候，可以多想一想是否有其它的解法。对于不同的解法，要注意比较，好在哪里，差在哪里，从各种不同解法的比较中作出合适的选择。

## 【例 4】装箱问题

有一个箱子容量为 v（正整数，0≤v≤20000），同时有 n 个物品（0<n≤30），每个物品有一个体积（正整数）。

要求从 n 个物品中，任取若干个装入箱内，使箱子的剩余空间为最小。

**输入：**

箱子的容量 v

物品数 n

接下来 n 行，分别表示这 n 个物品的体积

**输出：**

箱子剩余空间

**输入输出样例**

输入：　24
6
8
3
12
7
9
7

输出：　0

**【算法分析】**

**1. 使用回溯法计算箱子的最小剩余空间**

容量为 v 的箱子究竟应该装入哪些物品可使得剩余空间最小。显然在 n 个物品的体积和小于等于 v 的情况下，剩余空间为 v-n 个物品的体积和。但在 n 个物品的体积和大于 v 的情况下，没有一种可以直接找到问题解的数学方法。无奈之下，只能采用搜索的办法。设 a 和 s 为箱子的体积序列。其中 a[i]为箱子 i 的体积，s[i]为前 i 个箱子的体积和 $\sum_{k=1}^{i} a[k]$（1≤i≤n）；

best 为目前所有的装箱方案中最小的剩余空间。初始时 best=v；

确定搜索的几个关键因素：

**状态（k，v’）**：其中 k 为待装入箱子的物品序号，v’为箱子目前的剩余空间。

**目标 v’<best**：若箱子的剩余空间为目前为止最小，则 best 调整为 v’（best←v）；

**边界条件（v’-（s[n]-s[k-1]））≥best**：即便剩下的物品全部装入箱子(未装物品的体积和为 s[n]-s[k-1])，其剩余空间仍不小于 best，则放弃当前方案，回溯；

**搜索范围**：在未装入全部物品的前提下（k≤n），搜索两种可能情况：

若剩余空间装得下物品 k（v’≥a[k]），则物品 k 装入箱子，递归计算子状态（k+1，v’-a[k]）；

物品 k 不装入箱子，递归计算子状态（k+1，v’）；

我们用递归过程 search(k，v)描述这一搜索过程：

```
procedure search(k，v:integer);          {从状态(k，v)出发，递归计算最小剩余空间}
  begin
    if v<best then best←v;                    {若剩余空间为目前最小，则调整 best}
    if v-(s[n]-s[k-1])>=best    {若箱子即便装下全部物品，其剩余空间仍不小于 best，则回溯}
      then exit;
    if k<=n then                         {在未装入全部物品的前提下搜索两种可能情况}
       begin
         if v>=a[k]  {若剩余空间装得下物品 k，则物品 k 装入箱子，递归计算子状态}
              then search(k+1，v-a[k]);
          search(k+1，v);                     {物品 k 不装入箱子，递归计算子状态}
```

```
      end; {then}
    end; {search}
```

主程序如下：

```
    读箱子体积 v;
    读物品个数 n;
    s[0] ←0;                                          {物品装入前初始化}
    for i←1 to n do                                   {输入和计算箱子的体积序列}
    begin
      读第 i 个箱子的体积 a[i];
      s[i]←s[i-1]+a[i];
    end; {for}
    best←v;                                           {初始时，最小剩余空间为箱子体积}
    if s[n]<=v then best←v-s[n] {若所有物品能全部装入箱子，则剩余空间为问题解}
              else search(1, v);      {否则从物品 1 出发，递归计算最小剩余空间}
    输出最小剩余空间 best;
```

**2. 使用动态程序设计方法计算箱子的最小剩余空间**

如果按照物品序号依次考虑装箱顺序的话，则问题具有明显的阶段特征。问题是当前阶段的剩余空间最小，并不意味下一阶段的剩余空间也一定最小，即该问题并不具备最优子结构的特征。但如果将装箱的体积作为状态的话，则阶段间的状态转移关系顺其自然，可使得最优化问题变为判定性问题。设状态转移方程

f[i，j]——在前 i 个物品中选择若干个物品（必须包括物品 i）装箱，其体积正好为 j 的标志。显然 f[i，j]=f[i-1，j-box[i]]，即物品 i 装入箱子后的体积正好为 j 的前提是 f[i-1，j-box[i]]=true。初始时，f[0，0]=true（1≤i≤n，box[i]≤j≤v）。

由 f[i，j]=f[i-1，j-box[i]]可以看出，当前阶段的状态转移方程仅与上一阶段的状态转移方程相关。因此设 f0 为 i-1 阶段的状态转移方程，f1 为 i 阶段的状态转移方程，这样可以将二维数组简化成一维数组。我们按照下述方法计算状态转移方程 f1：

```
    fillchar(f0, sizeof(f0),0);                       {装箱前，状态转移方程初始化}
    f0[0]←true;
    for i←1 to n do                   {阶段 i:按照物品数递增的顺序考虑装箱情况}
      begin
        f1←f0;                                        {i 阶段的状态转移方程初始化}
        for j←box[i] to v do                          {状态 j：枚举所有可能的装箱体积}
          if f0[j-box[i]] then f1[j]←true;
                              {若物品 i 装入箱子后的体积正好为 j,则物品 i 装入箱子}
        f0←f1;                                        {记下当前装箱情况}
      end; {for}
```

经过上述运算，最优化问题转化为判定性问题。再借用动态程序设计的思想，计算装箱的最大体积 K=$\max\limits_{j=v\cdots 0}\{j \mid f[n,j]=true\}$。显然最小剩余空间为 v-k:

```
    for i←v downto 0 do                               {按照递减顺序枚举所有可能的体积}
      if f1[i] then
          begin     {若箱子能装入体积为 i 的物品，则输出剩余空间 v-i，并退出程序}
             writeln(v-i);  halt
          end; {then}
```

```
end.{for}
writeln(v);                    {在未装入一个物品的情况下输出箱子体积}
```

## 【例 5】合唱队形

**【问题描述】**

N 位同学站成一排，音乐老师要请其中的(N-K)位同学出列，使得剩下的 K 位同学排成合唱队形。

合唱队形是指这样的一种队形：设 K 位同学从左到右依次编号为 1，2，…，K，他们的身高分别为 $T_1$，$T_2$，…，$T_K$，则他们的身高满足 $T_1 < T_2 < \cdots < T_i$，$T_i > T_{i+1} > \cdots > T_K$ ($1 \leqslant i \leqslant K$)。

你的任务是，已知所有 N 位同学的身高，计算最少需要几位同学出列，可以使得剩下的同学排成合唱队形。

**【输入文件】**

输入文件 chorus.in 的第一行是一个整数 N（$2 \leqslant N \leqslant 100$），表示同学的总数。第一行有 n 个整数，用空格分隔，第 i 个整数 $T_i$（$130 \leqslant T_i \leqslant 230$）是第 i 位同学的身高（厘米）。

**【输出文件】**

输出文件 chorus.out 包括一行，这一行只包含一个整数，就是最少需要几位同学出列。

**【样例输入】**

8
186 186 150 200 160 130 197 220

**【样例输出】**

4

**【数据规模】**对于 50%的数据，保证有 $n \leqslant 20$；对于全部的数据，保证有 $n \leqslant 100$。

**【算法分析】**

如果不去洞悉其间蕴涵的数学规律，直接在穷举或搜索所有可能排列的基础是求解的话，时间复杂度为 $O(n*2^n)$，1 秒时限内仅能通过 $n \leqslant 20$ 范围内的测试数据。显然，这个算法的效率明显不符要求（），必须另辟新径。

### 解法 1：动态程序设计方法

我们按照由左而右和由右而左的顺序，将 n 个同学的身高排成数列。如何分别在这两个数列中寻求递增的、未必连续的最长子序列，就成为问题的关键。设

a 为身高序列，其中 a[i]为同学 i 的身高；

b 为由左而右身高递增的人数序列，其中 b[i]为同学 1‥同学 i 间（包括同学 i）身高满足递增顺序的最多人数。显然 b[i]= $\max\limits_{1 \le j \le i-1}$ {b[j]|同学 j 的身高<同学 i 的身高}+1；

c 为由右而左身高递增的人数序列，其中 c[i]为同学 n‥同学 i 间（包括同学 i）身高满足递增顺序的最多人数。显然 c[i]= $\max\limits_{i+1 \le j \le n}$ {c[j]|同学 j 的身高<同学 i 的身高}+1；

由上述状态转移方程可知，计算合唱队形的问题具备了最优子结构性质(要使 b[i]和 c[i]最大，子问题的解 b[j]和 c[k]必须最大($1 \leqslant j \leqslant i-1$，$i+1 \leqslant k \leqslant n$))和重迭子问题的性质(为求得 b[i]和 c[i]，必须一一查阅子问题的解 b[1]‥b[i-1]和 c[i+1]‥c[n])，因此可采用动态程序设计的方法求解。

显然，合唱队的人数为 $\max\limits_{1 \le i \le n}\{b[i]+c[i]\}-1$（公式中同学 i 被重复计算，因此减 1），n

减去合唱队人数即为解。具体算法如下：

```
  readln(n); {读学生数}
  for i:=1 to n do read(a[i]);                                  {读每个学生的身高}
  fillchar(b,sizeof(b),0);fillchar(c,sizeof(c),0);{身高满足递增顺序的两个队列初始化}
  for i:=1 to n do                                  {按照由左而右的顺序计算 b 序列}
   begin
       b[i]:=1;
       for j:=1 to i-1 do if (a[i]>a[j])and(b[j]+1>b[i]) then b[i]:=b[j]+1;
   end;{for}
  for i:=n downto 1 do                              {按照由右而左的顺序计算 c 序列}
   begin
    c[i]:=1;
       for j:=i+1 to n do if (a[j]<a[i])and(c[j]+1>c[i])then c[i]:=c[j]+1;
     end;{for}
  max:=0;                                {计算合唱队的人数 max(其中 1 人被重复计算)}
  for i:=1 to n do if b[i]+c[i]>max then max:=b[i]+c[i];
  writeln(n-max+1);                                                {输出出列人数}
```

这个算法的时间复杂度为 $0(n^2)$，在 1 秒时限内可解决 n≤100 范围内的问题。但是，这个算法并不是最优算法，还可以精益求精。

## 解法 2：二分法

设 x 为当前身高满足递增顺序的队列，其中 x[i]为第 i 高的队员身高；a 序列、b 序列和 c 序列的定义如解法 1。

设 x[i]的初始值为∞(1≤i≤n)，b[0]为 0。我们从同学 1 出发，按照由左而右的顺序计算身高递增的人数序列 b。在计算 b[i]时，通过二分法找出区间 x[1]‥x[i]中身高矮于同学 i 的元素个数 min（x 区间的左右指针为 min 和 max，中间指针为 mid)。寻找过程如下：

```
     min:=0;max:=i;{设 x 区间的左右指针}
   while min<max-1 do{若 x 区间存在，则通过二分法计算同学 i 前身高矮于同学 i 且满足递增顺序的最多人数 min}
   begin
     mid:=(min+max) div 2;{计算中间指针}
     if x[mid]<a[i]
     then min:=mid{搜索右区间}
     else max:=mid{搜索左区间}
   end;{while}
```

显然，b[i]=min+1，x[min+1]=a[i]。

在计算出 b 序列后，再采用类似方法由右而左计算身高递增的人数序列 c。最后得出出列人数为 n-$\max\limits_{1\le i\le n}\{b[i]+c[i]\}-1$（公式中同学 i 被重复计算，因此减 1)。具体算法如下：

```
  readln(n);{读学生数}
  for i:=1 to n do read(a[i]);{读每个学生的身高}
  for i:=1 to n do x[i]:=maxlongint;{身高满足递增顺序的队列初始化}
  b[0]:=0;
  for i:=1 to n do{由左而右计算 b 序列}
  begin
```

```
    min:=0;max:=i;{设 x 区间的左右指针}
    while min<max-1 do{若 x 区间存在，则通过二分法计算同学 i 前身高满足递增顺序的最多人数 min}
    begin
      mid:=(min+max) div 2;{计算中间指针}
      if x[mid]<a[i]
      then min:=mid{搜索右区间}
      else max:=mid{搜索左区间}
    end;{while}
    b[i]:=min+1;x[min+1]:=a[i]{同学 1‥同学 i 间（包括同学 i）最多有 min+1 个同学可排成身高递增的队列}
  end;{for}
  for i:=1 to n do x[i]:=maxlongint; {身高满足递增顺序的队列初始化}
  c[0]:=0;
  for i:=n downto 1 do{由右而左计算 c 序列}
  begin
    min:=0;max:=i;
    while min<max-1 do
    begin
      mid:=(min+max) div 2; 设 x 区间的左右指针}
      if x[mid]<a[i]
      then min:=mid{搜索右区间}
      else max:=mid{搜索左区间}
    end;{while}
    c[i]:=min+1;x[min+1]:=a[i] {同学 n‥同学 i 间（包括同学 i）最多有 min+1 个同学可排成身高递增的队列}
  end;{max}
  max:=0;{计算合唱队的人数 max（其中 1 人被重复计算）}
  for i:=1 to n do if b[i]+c[i]>max then max:=b[i]+c[i];
  writeln(n+1-max);{输出出列人数}
```

第二种解法的算法的时间复杂度为 O(n*log n)。事实证明，采用动态程序设计方法解题并不一定是最优的。由于第一种解法是通过顺序枚举的途径计算 b 序列和 c 序列的，而第二种解法采用了二分法计算，比第一种解法更精确地揭示了问题本质，冗余运算相对减少，因此其时效自然要好一些。

## 【例 6】橱窗布置（Flower）

**〖题目描述〗**

假设以最美观的方式布置花店的橱窗，有 F 束花，每束花的品种都不一样，同时，至少有同样数量的花瓶，被按顺序摆成一行，花瓶的位置是固定的，并从左到右，从 1 到 V 顺序编号，V 是花瓶的数目，编号为 1 的花瓶在最左边，编号为 V 的花瓶在最右边，花束可以移动，并且每束花用 1 到 F 的整数惟一标识，标识花束的整数决定了花束在花瓶中列的顺序即如果 I < J，则花束 I 必须放在花束 J 左边的花瓶中。

例如，假设杜鹃花的标识数为 1，秋海棠的标识数为 2，康乃馨的标识数为 3，所有的花束在放人花瓶时必须保持其标识数的顺序，即：杜鹃花必须放在秋海棠左边的花瓶中，秋海棠必须放在康乃馨左边的花瓶中。如果花瓶的数目大于花束的数目，则多余的花瓶必须空，即每个花瓶中只能放一束花。

每一个花瓶的形状和颜色也不相同，因此，当各个花瓶中放人不同的花束时会产生不同的美学效果，并以美学值(一个整数)来表示，空置花瓶的美学值为 0。在上述例子中，花瓶与

花束的不同搭配所具有的美学值，可以用如下表格表示。

根据表格，杜鹃花放在花瓶 2 中，会显得非常好看，但若放在花瓶 4 中则显得很难看。

为取得最佳美学效果，必须在保持花束顺序的前提下，使花的摆放取得最大的美学值，如果具有最大美学值的摆放方式不止一种，则输出任何一种方案即可。题中数据满足下面条件：1≤F≤100，F≤V≤100，－50≤$A_{IJ}$≤50，其中 AII 是花束 I 摆放在花瓶 J 中的美学值。输入整数 F，V 和矩阵($A_{IJ}$)，输出最大美学值和每束花摆放在各个花瓶中的花瓶编号。

| | 花瓶 1 | 花瓶 2 | 花瓶 3 | 花瓶 4 | 花瓶 5 |
|---|---|---|---|---|---|
| 杜鹃花 | 7 | 23 | -5 | -24 | 16 |
| 秋海棠 | 5 | 21 | -4 | 10 | 23 |
| 康乃馨 | -21 | 5 | -4 | -20 | 20 |

**1、假设条件**

1≤F≤100，其中 F 为花束的数量，花束编号从 1 至 F。

F≤V≤100，其中 V 是花瓶的数量。

-50≤$A_{ij}$≤50，其中 $A_{ij}$ 是花束 i 在花瓶 j 中的美学值。

**2、输入**

输入文件是 flower.in。

第一行包含两个数：F，V。

随后的 F 行中，每行包含 V 个整数，$A_{ij}$ 即为输入文件中第（i+1 ）行中的第 j 个数。

**3、输出**

输出文件必须是名为 flower.out 的正文文件，文件应包含两行：

第一行是程序所产生摆放方式的美学值。

第二行必须用 F 个数表示摆放方式，即该行的第 K 个数表示花束 K 所在的花瓶的编号。

**4、例子**

flower.in:

```
3 5
7 23 -5 -24 16
5 21 -4 10 23
-21 5 -4 -20 20
```

flower.out:

```
53
2 4 5
```

**【算法分析】**

问题实际就是给定 F 束花和 V 个花瓶，以及各束花放到不同花瓶中的美学值，要求你找出一种摆放的方案，使得在满足编号小的花放进编号小的花瓶中的条件下，美学值达到最大。

(1)将问题进行转化，找出问题的原型。首先，看一下上述题目的样例数据表格。

将摆放方案的要求用表格表现出来，则摆放方案需要满足：每行选且只选一个数(花瓶)；摆放方案的相邻两行中，下面一行的花瓶编号要大于上面一行的花瓶编号两个条件。这时可将问题转化为：给定一个数字表格，要求编程计算从顶行至底行的一条路径，使得这条路径所经过的数字总和最大(要求每行选且仅选一个数字)。同时，路径中相邻两行的数字，必须

保证下一行数字的列数大于上一行数字的列数。

看到经过转化后的问题，发现问题与例题 6 的数学三角形问题十分相似，数字三角形问题的题意是：

给定一个数字三角形，要求编程计算从顶至底的一条路径，使得路径所经过的数字总和最大(要求每行选且仅选一个数字)。同时，路径中相邻两行的数字，必须保证下一行数字的列数与上一行数字的列数相等或者等于上一行数字的列数加 1。

上例中已经知道：数字三角形中的经过数字之和最大的最佳路径，路径的每个中间点到最底层的路径必然也是最优的，可以用动态规划方法求解，对于“花店橱窗布置”问题经过转化后，也可采取同样的方法得出本题同样符合最优性原理。因此，可以对此题采用动态规划的方法。

(2)对问题原型动态规划方法的修改。“数字三角形”问题的动态规划方法为：已知它是用行数来划分阶段。假设用 a[i，j]表示三角形第 i 行的第 j 个数字，用 p [i，j]表示从最底层到 a[i,j]这个数字的最佳路径(路径经过的数字总和最大)的数字和，易得问题的动态转移方程为：

p[n+1,j]=0　　(1≤i≤n+1)

p[i,j]=max{p[i+1,j],p[i+1,j+1]}+a[i,j]

(1≤i≤i≤n，其中 n 为总行数)

分析两题的不同之处，就在于对路径的要求上。如果用 path[i]表示路径中第 i 行的数字编号，那么两题对路径的要求就是：“数字三角形”要求 path[i]≤path[i+1]≤path[i]+1，而本题则要求 path[i+1]>path[i]。

在明确两题的不同之后，就可以对动态规划方程进行修改了。假设用 b[i，j]表示美学值表格中第 i 行的第 j 个数字，用 q[i，j]表示从表格最底层到 b[i，j]这个数字的最佳路径(路径经过的数字总和最大)的数字和，修改后的动态规划转移方程为：

q[i,V+1]=-∞(1≤i≤F+1)

q[F，j]=b[F，j](1≤j≤V)

q[i，j]=max{q[i+1，k](j<K≤V+1)}+A[I,J](1≤I≤F，1≤J≤V)

这样，得出的max{q[1,k]}　(1≤j≤V)就是最大的美学值，算法的时间复杂度为 $O(FV^2)$。

(3)对算法时间效率的改进。先来看一下这样两个状态的求解方法：

q[i,j]=max{q[i+1，k] (j < k≤V+1)}+b[i，j]　(1≤i≤F，1≤j≤V)

q[i,j+1]=max{q[i+1，k] (j+1 < k≤V+1)}+a[i，j+1]　(1≤i≤F，　1≤j+1≤V)

上面两个状态中求 max{q[i+1，k]}的过程进行了大量重复的比较。此时对状态的表示稍作修改，用数组 t[i，j]=max{q[i,k](j≤k≤V+1)}(1≤i≤F，　1≤j≤V)表示新的状态。经过修改后，因为 q[i，j]=t[i+1，j+1]+a[i，j]，而 t[i，j]=max{t[i，j+1]，q[i,j](1≤i≤F，1≤j≤V)，所以得出新的状态转移方程：

t[i,V+1]=-∞　(1≤i≤F 十 1)

t[F，j]=max{t[F，j+1]，b[F，j]}　　(1≤j≤V)

t[i,j]=max{t[i，j+1]，t[i+1，j+1]+a[i，j]}　　(1≤i≤F，1≤j≤V)

这样，得出的最大美学值为 t[1,1]，新算法的时间复杂度为 O(F*V)，而空间复杂度也为 O(F*V)，完全可以满足 1≤F≤V≤100 的要求。下面给出这一问题的源程序。

**【参考程序】**

```
{$A+，B-，D+，E+，F-，G-，I+，L+，N-，O-，P-，Q-，R-，S+，T-，V+，X+}
{$M16384，0，655360}
program ex1;{花店橱窗布置问题}
    const st1='flower.in';      {输入文件名}
          st2='flower.out';     {输出文件名}
    var f,v:integer;  {f 为花束的数量；v 为花瓶的数量}
    b:array[1..100,1..100] of shortint;
                                  {b[i,j]为第 i 束花放进第 j 个花瓶的美学值}
```

```
    t:array[1..101,1..101] of integer;
         {t[i,¨为将第i到第f束花放进第j到第v个花瓶所可能得到的最大美学值}
    procedure readp;              {从文件中读入不同花束对应不同花瓶的美学值}
    var f1:text; i,j:integer;
    begin
      assign(f1,st1);reset(f1);
      readln (f1, f, v);
      for i:=1 to f do
        for j:=1 to v do
         read (f1, b [i, j]);
      close (f1);
    end;
  procedure main;             {用动态规划对问题求解}
   var i, j: integer;
   begin
      for i:=1 to f+1 do t[i, v+1]:=-9999;
      for j:=v downto 1 do
        if t[f,j+1]>b[f,j]
          then t[f,j]:=t[f, j+1]
          else t[f,j]:=b[f,j];{设定动态规划的初始条件,其中-9999表示负无穷}
      for i:=f-1 downto 1 do
        for j:= v downto 1 do
          begin
            t[i, j]:=t[i, j+1];
            if t[i+1,j+1] + b[i,j] > t[i, j] then
               t[i, j]:=t[i+1, j+1] +b[i, j];
          end;
   end;
  procedure print;                      {将最佳美学效果和对应方案输出到文件}
   var f1: text;
       i,j,p:integer;
          {为当前需确定位置的花束编号，p为第i束花应插入的花瓶编号的最小值}
   begin
     assign (f1, st2); rewrite (f1);
     writeln (f1, t[1,1]);
     p:=1;
     for i:=1 to f do
     begin                                {用循环依次确定各束花应插入的花瓶}
       j:=p;
       while t[i, j] =t[i, p] do inc (j);
       write (f1, j-1,' '); p:=j;
     end;
     writeln (f1);  close (f1);
  end;
begin
    readp;
    main;
    print;
```

end.

由此可看出，对于看似复杂的问题，通过转化就可变成简单的经典的动态规划问题。在问题原型的基础上，通过分析新问题与原问题的不同之处，修改状态转移方程，改变问题状态的描述和表示方式，就会降低问题规划和实现的难度，提高算法的效率。由此可见，动态规划问题中具体的规划方法将直接决定解决问题的难易程度和算法的时间与空间效率，而注意在具体的规划过程中的灵活性和技巧性将是动态规划方法提出的更高要求。

## 【例 7】方格取数

设有 n*n 的方格图（N≤8），我们将其中的某些方格中填入正整数，而其他的方格中则放入数字 0 。如图 1 1.2.5 所示（见样例）：

向右

向下

| A | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 13 | 0 | 0 | 6 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 7 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 14 | 0 | 0 | 0 | 0 |
| 5 | 0 | 21 | 0 | 0 | 0 | 4 | 0 | 0 |
| 6 | 0 | 0 | 15 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 14 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

B

图 1 1.2.5

某人从图的左上角的 A 点出发，可以向下行走，也可以向右走，直到到达右下角的 B 点。在走过的路上，它可以取走方格中的数（取走后的方格中将变为数字 0 ）。此人从 A 点到 B 点共走两次，试找出 2 条这样的路径，使得取得的数之和最大。

**【输入格式】**

输入的第一行为一个整数 N（表示 N ＊ N 的方格图），接下来的每行右三个整数，前两个表示位置，第三个数为该位置上所放的数。一行单独的 0 表示输入结束。

**【输出格式】**

只需输出一个整数，表示 2 条路径上取得的最大的和

**【样例输入】**

```
8
2 3 13
2 6 6
3 5 7
4 4 14
5 2 21
5 6 4
6 3 15
7 2 14
0 0 0
```

**【样例输出】**

67

【算法分析】

我们对这道题并不陌生。如果求一条数和最大的路径，读者自然会想到动态程序设计方法。现在的问题是，要找出这样的两条路径，是否也可以采用动态程序设计方法呢？回答是可以的。

1、状态的设计

对于本题来说，状态的选定和存储对整个问题的处理起了决定性的作用。

我们从（1，1）出发，每走一步作为一个阶段，则可以分成 2*n-1 个阶段：

第一个阶段，两条路径从（1，1）出发；

第二个阶段，两条路径可达（2，1）和（1，2）；

第三个阶段，两条路径可达的位置集合为（3，1）、（2，2）和（1，3）；

……

第 2*n-1 个阶段，两条路径汇聚（n，n）；

在第 k(1≤k≤2*n-1)个阶段，两条路径的终端坐标（$x_1$，$y_1$）（$x_2$，$y_2$）位于对应的右下对角线上。如图 1 1.2.6 所示：

**如果我们将两条路径走第 i 步的所有可能位置定义为当前阶段的状态的话，面对的问题就是如何存储状态了。方格取数问题的状态数目十分庞大，每一个位置是两维的，且又是求两条最佳路径，这就要求在存储上必须做一定的优化后才有可能实现算法的程序化。**

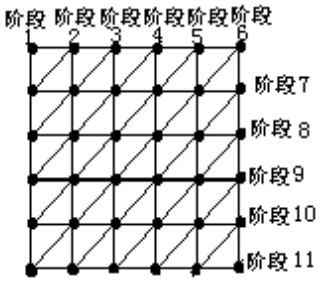



图 1 1.2.6

**主要的优化就是：舍弃一切不必要的存储量。**为此，我们取位置中的 X 坐标（$x_1$，$x_2$）作状态，其中

$$(1\leqslant x_1\leqslant k)\wedge(x_1\in\{1..n\})\wedge(1\leqslant x_2\leqslant k)\wedge(x_2\in\{1..n\})$$

直接由 X 坐标计算对应的 Y 坐标：

$$(y_1=k+1-x_1)\wedge(y_1\in\{1..n\})\wedge(y_2=k+1-x_2)\wedge(y_2\in\{1..n\})$$

2.状态转移方程

设两条路径在 k 阶段的状态为（$x_1$，$x_2$）。由（$y_1=k+1-x_1$）∧（$y_1\in\{1..n\}$）得出第一条路径的坐标为（$x_1$，$y_1$）；由（$y_2=k+1-x_2$）∧（$y_2\in\{1..n\}$）得出第二条路径的坐标为（$x_2$，$y_2$）。假设在 k-1 阶段，两条路径的状态为（$x_1'$，$x_2'$）且（$x_1'$，$x_2'$）位于（$x_1$，$x_2$）状态的左邻或下邻位置。因此我们设条路径的延伸方向为 $d_1$和 $d_2$：$d_i=0$，表明第 i 条路径由（$x_i'$，$y_i'$）向右延伸至（$x_i$，$y_i$）；$d_i=1$，表明第 i 条路径由（$x_i'$，$y_i'$）向下延伸至（$x_i$，$y_i$）（1≤i≤2）。显然（$x_1'=x_2'$）∧（$d_1=d_2$），表明两条路径重合，同时取走了（$x_1'$，$y_1'$）和（$x_1$，$y_1$）中的数，这种取法当然不可能得到最大数和的。

分析两种可能：

⑴若 $x_1=x_2$，则两条路径会合于 $x_1$ 状态，可取走（$x_1$，$y_1$）中的数(图 11.2.6)；

$x_1$’（$x_2$’）→$x_1=x_2$

↑

$x_2$’（$x_1$’）

图 11.2.6

⑵若 $x_1\neq x_2$，则在 k 阶段，第一条路径由 $x_1$’状态延伸至 $x_1$ 状态，第二条路径由 $x_2$’状态延伸至 $x_2$ 状态，两条路径可分别取走（$x_1$，$y_1$）和（$x_2$，$y_2$）中的数(图 11.2.7)；

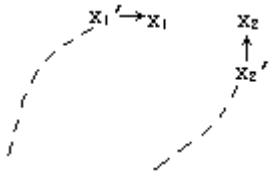



图 11.2.7

设

f[k，$x_1$，$x_2$]—在第 k 阶段，两条路径分别行至 $x_1$ 状态和 $x_2$ 状态的最大数和。显然

**k=1 时，f[1，1，1]=0；**

**k≥2 时，f[k，$x_1$，$x_2$]=max{f[k-1， $x_1$’，$x_2$’]+($x_1$，$y_1$)的数字| $x_1=x_2$**

**f[k-1， $x_1$’，$x_2$’]+($x_1$，$y_1$)的数字+($x_2$，$y_2$)的数字| $x_1\neq x_2$}**

**1≤k≤2*n-1，$x_1$’∈可达 $x_1$ 的状态集合，$x_2$’∈可达 $x_2$ 的状态集合，（$x_1$’≠$x_2$’）∨($d_1\neq d_2$)；**

上述状态转移方程符合最优子结构和重叠子问题的特性，因此适用于动态程序设计方法求解。由于第 k 个阶段的状态转移方程仅与第 k-1 个阶段的状态发生联系，因此不妨设

$f_0$—第 k-1 个阶段的状态转移方程；

$f_1$—第 k 个阶段的状态转移方程；

初始时，$f_0$[1，1]=0。经过 2*n-1 个阶段后得出的 $f_0$[n，n]即为问题的解。

**3. 多进程决策的动态程序设计**

由于方格取数问题是对两条路径进行最优化决策的，因此称这类动态程序设计为分阶段、多进程的最优化决策过程。设

阶段 i：准备走第 i 步（1≤i≤2*n-1）；

状态（$x_1$’，$x_2$’）： 第 i-1 步的状态号（1≤$x_1$’，$x_2$’≤i-1。$x_1$’，$x_2$’∈{1..n}）

决策（$d_1$，$d_2$）：第一条路径由 $x_1$’状态出发沿 $d_1$ 方向延伸、第二条路径由 $x_2$’状态 出发沿 $d_2$ 方向延伸，可使得两条路径的数和最大（0≤$d_1$，$d_2$≤1。方向 0 表示右移，方向 1 表示下移）；

具体计算过程如下：

```
fillchar(f0, sizeof(f0), 0);                    { 行走前的状态转移方程初始化}
f0[1, 1]←0;
for i←2 to n+n-1 do                             {阶段：准备走第 i 步}
  begin
  fillchar(f1, sizeof(f1), 0);                  { 走第 i 步的状态转移方程初始化}
  for x1’←1 to i-1 do                           {枚举两条路径在第 i-1 步时的状态 x1’和 x2’}
    for x2’←1 to i-1 do
     begin
```

```
        计算 y1' 和 y2';
        if (x1', y1')和(x2', y2')在界内 then
         for d1←0 to 1 do                    {决策：计算两条路径的最佳延伸方向}
          for d2←0 to 1 do
           begin
            第 1 条路径沿 d1 方向延伸至(x1, y1);
            第 2 条路径沿 d2 方向延伸至(x2, y2);
            if ((x1, y1)和(x2, y2)在界内)∧((d1≠d2)∨(x1≠x2)) {计算第 i 步的状态转移方程}
               then f1[x1, x2]←max{f0[x1', x2']+map[x1, y1] | x1=x2, f0[x1',
x2']+map[x1, y1]+map[x2, y2] | x1≠x2}
           end; {for}
        end; {for}
       f0←f1;                                {记下第 i 步的状态转移方程}
     end; {for}
   输出两条路径取得的最大数和 f0[n, n];
```

# 【上机练习】

**1、数塔问题**(tower.pas)

有形如图 1 所示的数塔，从顶部出发，在每一结点可以选择向左走或是向右走，一起走到底层，要求找出一条路径，使路径上的值最大。

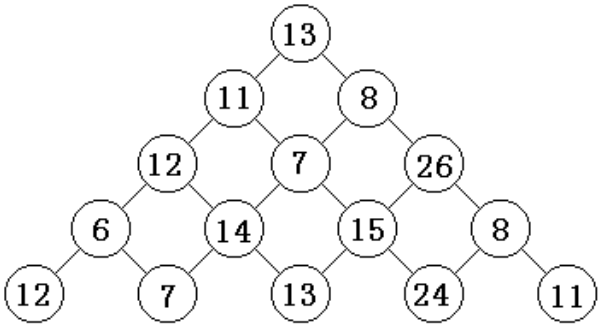



图 1

**【输入输出样例】**

输入：

```
5
13
11   8
12   7    26
6   14    15    8
12   7    13   24    11
```

输出：

max=86

**2、拦截导弹**(missile.pas)

某国为了防御敌国的导弹袭击，发展出一种导弹拦截系统。但是这种导弹拦截系统有一个缺陷：虽然它的第一发炮弹能够到达任意的高度，但是以后每一发炮弹都不能高于前一发的高度。某天，雷达捕捉到敌国的导弹来袭。由于该系统还在试用阶段，所以只有一套系统，因此有可能不能拦截所有的导弹。

输入导弹依次飞来的高度（雷达给出的高度数据是不大于 30000 的正整数），计算这套系统最多能拦截多少导弹，如果要拦截所有导弹最少要配备多少套这种导弹拦截系统。

**【输入输出样例】**

输入：

389 207 155 300 299 170 158 65

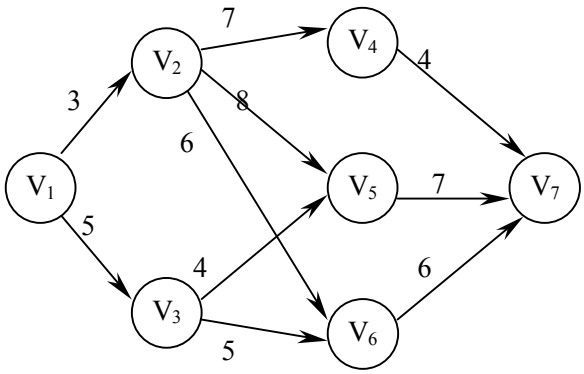
输出：

6（最多能拦截的导弹数）

2（要拦截所有导弹最少要配备的系统数）

**3、最短路径（short.pas）**

在下图中找出从起点到终点的最短路径。




[样例输入]

```
short.in
7
0 3 5 0 0 0 0
0 0 0 7 8 6 0
0 0 0 0 4 5 0
0 0 0 0 0 0 4
0 0 0 0 0 0 7
0 0 0 0 0 0 6
0 0 0 0 0 0 0
```

[样例输出]

```
short.out
minlong=14
1  2  4  7
```

**4、骑士游历问题(knight.pas)**

**5、砝码称重(weight.pas)**

**6、数字游戏（Game.pas）**

**7、装箱问题(boxes.pas)**

**8、合唱队形（Chorus.pas）**

**9、橱窗布置（Flower.pas）**

**10、方格取数(pane.PAS)**

# 第四节　背包问题

## 【例 1】0/1 背包

**【问题描述】**

一个旅行者有一个最多能用 m 公斤的背包，现在有 n 件物品，它们的重量分别是 W1，W2，...,Wn,它们的价值分别为 C1,C2,...,Cn.若每种物品只有一件求旅行者能获得最大总价值。

**【输入格式】**

第一行：两个整数，M(背包容量，M<=200)和 N(物品数量，N<=30)；

第 2..N+1 行：每行二个整数 Wi,Ci，表示每个物品的重量和价值。

**【输出格式】**

仅一行，一个数，表示最大总价值。

**【样例输入】**

package.in

```
10 4
2  1
3  3
4  5
7  9
```

**【样例输出】**

package.out

```
12
```

## 【解法一：动态规划】

**【算法分析】**

显然这个题可用深度优先方法对每件物品进行枚举(选或不选用 0,1 控制)。程序简单,但是当 n 的值很大的时候不能满足时间要求，时间复杂度为 $O(2^n)$。按递归的思想我们可以把问题分解为子问题,使用递归函数 。

设 f(i，x)表示前 i 件物品，总重量不超过 x 的最优价值

则 f（i，x）=max(f(i－1，x-W[i])+C[i]，f（i－1，x））

f（n，m）即为最优解，边界条件为 f（0，x）＝0 ，f(i,0)=0;

**下面例出 F[I，X]的值，I 表示前 I 件物品，X 表示重量**

| | F[I,1] | F[I,2] | F[I,3] | F[I,4] | F[I,5] | F[I,6] | F[I,7] | F[I,8] | F[I,9] | F[I,10] |
|---|---|---|---|---|---|---|---|---|---|---|
| I=1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| I=2 | 0 | 1 | 3 | 3 | 4 | 4 | 4 | 4 | 4 | 4 |
| I=3 | 0 | 1 | 3 | 5 | 5 | 6 | 8 | 8 | 9 | 9 |
| I=4 | 0 | 1 | 3 | 5 | 5 | 6 | 9 | 9 | 10 | 12 |

动态规划方法(顺推法):

**【参考程序】**

```
program package;
 const   maxm=200;maxn=30;
```

```
 type ar=array[1..maxn] of integer;
 var
    m,n,j,i:integer;
    c,w:ar;
    f:array[0..maxn,0..maxm] of integer;

 function max(x,y:integer):integer;
 begin
  if x>y then max:=x else max:=y;
 end;

BEGIN
  assign(input,' package.in');
  assign(output,' package.out');
  reset(input); rewrite(output);
  readln(m,n);
  for i:= 1 to n do
     readln(w[i],c[i]);
  for i:=1 to m do f[0,i]:=0;
  for i:=1 to n do f[i,0]:=0;
  for i:=1 to n do
   for j:=1 to m do
    begin
      if j>=w[i]  then f[i,j]:=max(f[i-1,j-w[i]]+c[i],f[i-1,j])
                  else f[i,j]:=f[i-1,j];
    end;
  writeln(f[n,m]);
  close(input);
  close(output);
END.
```

使用二维数组存储各子问题时方便，但当 maxm 较大时如 maxn=2000 时不能定义二维数组 f,怎么办,其实可以用一维数组,但是上述中 j:=1 to m 要改为 j:=m downto 1,为什么?请大家自己解决。

## 【解法二：深搜算法】

### 【算法分析】

设 n 件物品的重量分别为 $w_1,w_2,\cdots,w_n$;，物品的价值分别为 $v_1,v_2,\cdots,v_n$。采用递归寻找物品的选择方案。设前面已有了多种选择的方案，并保留了其中总价值最大的方案于数组 result 中，该方案的总价值存于变量 maxv。当前正在考察某一新的方案，其物品选择情况保存于数组 option 中。假定当前方案已考虑了前 i-1 件物品，现在要考虑第 i 件物品；当前方案已包含的物品的重量之和为 tw；至此，若其余物品都选择是可能的话，本方案能达到的总价值的期望值设为 tv。算法引入 tv 是当一旦当前方案的总价值的期望值也小于前面方案的总价值 maxv 时，继续考察当前方案变成无意义的工作，应终止当前方案，立即去考察下一个方案。因为当方案的总价值不比 maxv 大时，该方案不会再被考察。这同时保证后面找到的方案一定会比前面的方案更好。

对于第 i 件物品的选择有两种可能：

（1）物品 i 被选择，这种可能性仅当包含它不会超过方案总重量的限制时才是可行的。

选中后，继续递归去考虑其余物品的选择；

（2）物品 i 不被选择，这种可能性仅当不包含物品 i 也有可能会找到价值更大的方案的情况。

按以上思想写出递归算法如下：

【算法】背包问题，找最佳方案

```
try (物品 i,当前选择已达到的重量和 tw,本方案可能达到的总价值为 tv)
begin
    {考虑物品 i 包含在当前方案中的可能性}
    if 包含物品 i 是可接受的 then
    begin
         将物品 i 包含在当前方案中；
         if i＜n then try(i＋1,tw 十物品 i 的重量,tv);
            else {又一个完整方案，因它比前面的方案好，以它作为最佳方案}
                  以当前方案作为临时最佳方案保存
         恢复物品 i 不包含状态；
    end;
    {考虑物品 i 不包含在当前方案中的可能性}
     if 不包含物品 i 仅是可考虑的 then
        if i＜n then try(i＋1, tw, tv-物品 i 的价值);
           else {又一个完整方案，因它比前面的方案好，以它作为最佳方案。}
                 以当前方案作为临时最佳方案保存；
end;
```

按上述算法编写函数和程序如下：

**【参考程序】**

```
const maxn=20;
var i,n,limitw,maxv,totalv:longint;
    w,v:array[1..maxn] of longint;
    result,option:array[1..maxn] of boolean;
procedure try(i,tw,tv:longint);
var k:longint;
begin
     if tw+w[i]<=limitw then
     begin
          option[i]:=true;
          if i<n then try(i+1,tw+w[i],tv)
                 else begin for k:=1 to n do result[k]:=option[k];
                            maxv:=tv end;
          option[i]:=false
     end;
     if tv-v[i]>maxv then
        if i<n then try(i+1,tw,tv-v[i])
               else begin for k:=1 to n do result[k]:=option[k];
                          maxv:=tv-v[i] end
end;
```

```
BEGIN
     write('输入物品种数 n:'); readln(n);
     writeln('输入各物品的重量和价值:');
     totalv:=0;
     for i:=1 to n do
     begin
          write('Input w[',i,'],v[',i,']:');
          readln(w[i],v[i]);
          totalv:=totalv+v[i]
     end;
     write('输入限制重量 limitw:'); readln(limitw);
     maxv:=0;
     for i:=1 to n do option[i]:=false;
     try(1,0,totalv);
     write('选择方案为:');
     for i:=1 to n do if result[i] then write(i,' ');
     writeln;
     writeln('总价值为:',maxv)
END.
```

## 【例 2】完全背包

**【问题描述】**

设有 n 种物品，每种物品有一个重量及一个价值。但每种物品的数量是无限的，同时有一个背包，最大载重量为 M，今从 n 种物品中选取若干件(同一种物品可以多次选取)，使其重量的和小于等于 M，而价值的和为最大。

**【输入格式】**

第一行：两个整数，M(背包容量，M<=200)和 N(物品数量，N<=30)；

第 2..N+1 行：每行二个整数 Wi,Ui，表示每个物品的重量和价值。

**【输出格式】**

仅一行，一个数，表示最大总价值。

**【样例一输入】**

knapsack.in

```
10 4
2  1
3  3
4  5
7  9
```

**【样例一输出】**

knapsack.out

```
max=12
no.1   weight: 2   worth: 1   get: 0
no.2   weight: 3   worth: 3   get: 1
no.3   weight: 4   worth: 5   get: 0
no.4   weight: 7   worth: 9   get: 1
```

**【样例二输入】**

knapsack.in

```
12 4
2  1
3  3
4  5
7  9
```

**【样例二输出】**

knapsack.out

```
max=15
no.1   weight: 2   worth: 1   get: 0
no.2   weight: 3   worth: 3   get: 0
no.3   weight: 4   worth: 5   get: 3
no.4   weight: 7   worth: 9   get: 0
```

## 【解法一】

**【算法分析】**

• 记 W(1)，W(2)，…，W(N)每种物品的重量，U(1)，U(2)，…，U(N)为每种物品的价值，X(1)，X(2)，…，X(N)为每种物品选取的数量，问题成为满足：

条件 Σ X(I)W(I)≤M 且 Σ X(I)U(I) 为最大。

• 记 FK（Y）为取前 K 种物品，限制重量为 Y 时取得价值最大，则：

F0(Y)=0，即对一切 Y，一件物品都不取时，最大价值为 0；

FK(0)=0，即最大重量限制为 0 时，不能取得任何物品，所以最大价值也为 0；

F1(Y)=[y/w1]u1，即仅取第一种物品，则最大价值为尽可能多的装第一种物品，所以能装的数量为[y/w1]，而得到的价值为[y/w1]u1。

一般公式:FK(Y)=max{ FK-1(Y)，FK(Y-WK)+UK},并约定，当 Y<0 时，FK(Y)=负无穷大。

下面用一个具体的例子来说明求解的过程。

M=10, N=4

W1=2，W2=3，W3=4，W4=7

U1=1，U2=3，U3=5，U4=9

下面列出 FK(Y)：

| X \ Y | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 2 | 2 | 3 | 3 | 4 | 4 | 5 | 注 1 |
| 2 | 0 | 1 | 3 | 3 | 4 | 6 | 6 | 7 | 9 | 9 | 注 2 |
| 3 | 0 | 1 | 3 | 5 | 5 | 6 | 8 | 10 | 10 | 11 | 注 3 |
| 4 | 0 | 1 | 3 | 5 | 5 | 6 | 9 | 10 | 10 | 12 | 注 4 |

注 1・第 1 行表示 k=1，即取第一种物品，当 Y=1 时，无法取，所以 F1(1)=0，当 y=2 时，可取 1 个第一种物品，价值为 1，…，当 y=10(即 M)，此时，可取 5 个第一种物品，价值为 5。

注 2・当可以取 2 种物品时。

当 y=1 时，为 0(什么都不能取)；

当 y=2 时，仅能取第一种物品，所以 F2(2)=1；

当 y=3 时，有 2 种取法。取一个第一种物品，价值为 1；取一个第二种物品，价值为 3，取大者，所以 F2(3)=3。

当 y=4 也有 2 种取法，取第一种物品 2 件，价值为 2；取第二种物品 1 件价值为 3，所以 F2(4)=3。

当 y=5 时，有 2 种取法，取第一种物品 2 件，价值为 2；取第一种物品，第二种物品各 1 件，价值为 4，所以 F2(5)=4。以此类推。计算 F2(10)时，有 2 种考虑，第一种是全部取第一种，即 F1(10)=5；第二种是由 F2(7)+3=9，取大者，得到 F2(10)=9。

注 3・

注 4・F4(10)的计算方法为:

上面给出的是计算 F(I，J)的方法，但是还不能确定每种物品的选取个数。下面我们用倒推法来求出每种物品的个数。记 F(N，M)为目标，检查:F(N-1，M)与 F(N，M-WN)+UN，若前者大，则无输出，由 F(N，M)-> F(N-1，M)。若后者大，则输出 WN，计算 F(N，M-WN)。当 N-1，N-2，…，到达 1 时，则全部输出。

**【参考程序】:（顺推法）**

```
Program knapsack;
  Const  maxm=200;maxn=30;
  type
       tlist=array[1..maxn] of byte;
       tmake=array[0..maxn,0..maxm] of integer;
var i,n,m:integer;
    w,u:tlist;
    f:tmake;
procedure make;
  var i,j:byte;
begin
  for i:=1 to n do
  begin
    for j:=1 to w[i]-1 do
      f[i,j]:=f[i-1,j];
    for j:=w[i] to m do
    if f[i-1,j]>f[i,j-w[i]]+u[i] then   f[i,j]:=f[i-1,j]
                        else  f[i,j]:=f[i,j-w[i]]+u[i];
  end;
end;

procedure print;
var get:tlist;
   i,j:byte;
```

```
begin
  fillchar(get,sizeof(get),0);
  i:=n; j:=m;
    while i>0 do
       if f[i,j]=f[i-1,j] then dec(i)
         else begin
                dec(j,w[i]);
                inc(get[i]) ;
              end;
    writeln('max=', f[n,m]);
    for i:=1 to n do
      writeln('no.',i,'  weight:',w[i]:2,'  worth:',u[i]:2,'  get:',get[i]:2);
end;
BEGIN
  assign(input,'knapsack.in');
  assign(output,'knapsack.out');
  reset(input); rewrite(output);
  fillchar(w,sizeof(w),0);
  fillchar(u,sizeof(u),0);
  readln(m,n);
  for i:=1 to n do
       read(w[i],u[i]);
  make;
  print; {或者 writeln('max=', f[n,m]);}
  close(input); close(output);
END.
```

## 【解法二】

**【算法分析】**

本问题的数学模型如下：

设 f(x)表示重量不超过 x 公斤的最大价值， 则 f(x)=max{f(x-w[i])+u[i]}

当 x>=w[i] 1<=i<=n

**【样例一】中间结果**

**下面例出 F[X]表示重量不超过 x 公斤的最大价值，X 表示重量**

| | F[1] | F[2] | F[3] | F[4] | F[5] | F[6] | F[7] | F[8] | F[9] | F[10] |
|---|---|---|---|---|---|---|---|---|---|---|
| X=1 | 0 | | | | | | | | | |
| X=2 | 0 | 1 | | | | | | | | |
| X=3 | 0 | 1 | 3 | | | | | | | |
| X=4 | 0 | 1 | 3 | 5 | | | | | | |
| X=5 | 0 | 1 | 3 | 5 | 5 | | | | | |
| X=6 | 0 | 1 | 3 | 5 | 5 | 6 | | | | |
| X=7 | 0 | 1 | 3 | 5 | 5 | 6 | 9 | | | |
| X=8 | 0 | 1 | 3 | 5 | 5 | 6 | 9 | 10 | | |
| X=9 | 0 | 1 | 3 | 5 | 5 | 6 | 9 | 10 | 10 | |
| X=10 | 0 | 1 | 3 | 5 | 5 | 6 | 9 | 10 | 10 | 12 |

【样例二】中间结果

下面例出 F[X]表示重量不超过 x 公斤的最大价值，X 表示重量

| | F[1] | F[2] | F[3] | F[4] | F[5] | F[6] | F[7] | F[8] | F[9] | F[10] | F[11] | F[12] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| X=1 | 0 | | | | | | | | | | | |
| X=2 | 0 | 1 | | | | | | | | | | |
| X=3 | 0 | 1 | 3 | | | | | | | | | |
| X=4 | 0 | 1 | 3 | 5 | | | | | | | | |
| X=5 | 0 | 1 | 3 | 5 | 5 | | | | | | | |
| X=6 | 0 | 1 | 3 | 5 | 5 | 6 | | | | | | |
| X=7 | 0 | 1 | 3 | 5 | 5 | 6 | 9 | | | | | |
| X=8 | 0 | 1 | 3 | 5 | 5 | 6 | 9 | 10 | | | | |
| X=9 | 0 | 1 | 3 | 5 | 5 | 6 | 9 | 10 | 10 | | | |
| X=10 | 0 | 1 | 3 | 5 | 5 | 6 | 9 | 10 | 10 | 12 | | |
| X=11 | 0 | 1 | 3 | 5 | 5 | 6 | 9 | 10 | 10 | 12 | 14 | |
| X=12 | 0 | 1 | 3 | 5 | 5 | 6 | 9 | 10 | 10 | 12 | 14 | 15 |

【参考程序】：（顺推法）

```
program knapsack04;
 const  maxm=200;maxn=30;
 type   ar=array[0..maxn] of integer;
 var
    m,n,j,i,t:integer;
    u,w:ar;
    f:array[0..maxm] of integer;
BEGIN
  assign(input,'knapsack.in');
  assign(output,'knapsack.out');
  reset(input); rewrite(output);
  readln(m,n);
  for i:= 1 to n do
     readln(w[i],u[i]);
  f[0]:=0;
  for i:=1 to m do
   for j:=1 to n do
    begin
       if i>=w[j]  then t:=f[i-w[j]]+u[j];
       if t>f[i] then f[i]:=t
    end;
  writeln( 'max=' ,f[m]);
  close(input);
  close(output);
END.
```

# 【上机练习】

## 1、0/1 背包

**【问题描述】**

一个旅行者有一个最多能用 m 公斤的背包，现在有 n 件物品，它们的重量分别是 W1，W2，...,Wn,它们的价值分别为 C1,C2,...,Cn。若每种物品只有一件，求旅行者能获得最大总价值。

**【输入格式】**

第一行：两个整数，M(背包容量，M<=200)和 N(物品数量，N<=30)；

第 2..N+1 行：每行二个整数 Wi,Ci，表示每个物品的重量和价值。

**【输出格式】**

仅一行，一个数，表示最大总价值。

**【样例输入】**

package.in

```
10 4
2  1
3  3
4  5
7  9
```

**【样例输出】**

package.out

```
12
```

## 2、完全背包

**【问题描述】**

设有 n 种物品，每种物品有一个重量及一个价值。但每种物品的数量是无限的，同时有一个背包，最大载重量为 M，今从 n 种物品中选取若干件(同一种物品可以多次选取)，使其重量的和小于等于 M，而价值的和为最大。

**【输入格式】**

第一行：两个整数，M(背包容量，M<=200)和 N(物品数量，N<=30)；

第 2..N+1 行：每行二个整数 Wi,Ui，表示每个物品的重量和价值。

**【输出格式】**

仅一行，一个数，表示最大总价值。

**【样例输入】**

knapsack.in

```
10 4
2  1
3  3
4  5
7  9
```

**【样例输出】**

knapsack.out

max=12

## 3、货币系统(Money Systems)

**【问题描述】**

母牛们不但创建了他们自己的政府而且建立了自己的货币系统。他们对货币的数值感到好奇。传统地，一个货币系统是由 1，5，10，20 或 25，50，100 的单位面值组成的。母牛想知道用货币系统中的货币来构造一个确定的面值，有多少种不同的方法。

举例来说，使用一个货币系统 {1,2,5,10,...}产生 18 单位面值的一些可能的方法是:18x1，9x2，8x2+2x1，3x5+2+1 等等其它。

写一个程序来计算有多少种方法，用给定的货币系统来构造一个确定的面值。

**【输入格式】**

货币系统中货币的种类数目是 V。 (1<= V<=25)

要构造的面值是 N。 (1<= N<=10,000)

| | |
|---|---|
| 第 1 行: | 二个整数，V 和 N |
| 第 2 ..V+1 行： | 可用的货币 V 个整数 (每行一个)。 |

**【输出格式】**

单独的一行包含那个可能的构造的方案数。

**【样例输入】**

money.in

```
3 10
1
2
5
```

**【样例输出】**

money.out

```
10
```

## 4、竞赛总分(Score Inflation)

**【问题描述】**

学生在我们 USACO 的竞赛中的得分越多我们越高兴。我们试着设计我们的竞赛以便人们能尽可能的多得分。

现在要进行一次竞赛，总时间 T 固定，有若干类型可选择的题目，每种类型题目可选入的数量不限，每种类型题目有一个 si(解答此题所得的分数)和 ti(解答此题所需的时间)，现要选择若干题目，使解这些题的总时间在 T 以内的前提下，所得的总分最大。

输入包括竞赛的时间,M(1 <= M <= 10000)和题目类型数目 N(1 <= N <= 10000)。

后面的每一行将包括两个整数来描述一种"题型":

第一个整数说明解决这种题目能得的分数(1 <= points <= 10000),第二整数说明解决这种题目所需的时间(1 <= minutes <= 10000)。

**【输入格式】**

| | |
|---|---|
| 第 1 行: | 两个整数：竞赛的时间 M 和题目类型数目 N。 |
| 第 2-N+1 行: | 两个整数：每种类型题目的分数和耗时。 |

**【输出格式】**

单独的一行，在给定固定时间里得到的最大的分数。

**【样例输入】**

inflate.in

300 4

100 60

250 120

120 100

35 20

**【样例输出】**

inflate.out

605 {从第 2 种类型中选两题和第 4 种类型中选三题}

## 5、质数和分解（PRIME.PAS）

**【问题描述】**

任何大于 1 的自然数 N，都可以写成若干个大于等于2且小于等于 N 的质数之和表达式(包括只有一个数构成的和表达式的情况)，并且可能有不止一种质数和的形式。例如9 的质数和表达式就有四种本质不同的形式：9 = 2+5+2 = 2+3+2+2 = 3+3+3 = 2+7 。

这里所谓两个本质相同的表达式是指可以通过交换其中一个表达式中参加和运算的各个数的位置而直接得到另一个表达式。

试编程求解自然数 N 可以写成多少种本质不同的质数和表达式。

**【输入文件】（PRIME.IN）：**

文件中的每一行存放一个自然数 N ， 2≤N≤200。

**【输出文件】（PRIME.OUT）：**

依次输出每一个自然数 N 的本质不同的质数和表达式的数目。

**样例一：**

PRIME.IN

2

PRIME.OUT

1

**样例二：**

PRIME.IN

200

PRIME.OUT

9845164

# 第五节 动态规划应用举例

## 例 1、挖地雷（Mine.pas）

**【问题描述】**

在一个地图上有 N 个地窖（N<=200）,每个地窖中埋有一定数量的地雷。同时，给出地窖之间的连接路径，并规定路径都是单向的。某人可以从任一处开始挖地雷，然后沿着指出的连接往下挖（仅能选择一条路径），当无连接时挖地雷工作结束。设计一个挖地雷的方案，使他能挖到最多的地雷。

**【输入格式】**

N {地窖的个数}

$W_1$，$W_2$，……$W_N$ {每个地窖中的地雷数}

X1，Y1 {表示从 X1 可到 Y1}

X2，Y2

……

0，0 {表示输入结束}

**【输出格式】**

K1——K2——……——Kv {挖地雷的顺序}

MAX {最多挖出的地雷数}

**【输入样例】**(Mine.in)

```
6
5 10 20 5 4 5
1 2
1 4
2 4
3 4
4 5
4 6
5 6
0 0
```

**【输出样例】**(Mine.out)

```
3-4-5-6
34
```

**【算法分析】**

本题是一个经典的动态规划问题。很明显，题目规定所有路径都是单向的，所以满足无后效性原则和最优化原理。设 W（i）为第 i 个地窖所藏有的地雷数，A（i,j）表示第 i 个地窖与第 j 个地窖之间是否有通路，F（i）为从第 i 个地窖开始最多可以挖出的地雷数，则有如下递归式：

F（i）=MAX { F(j) + W(i) } (i<j<=n , A(i,j)=1)

边界：F（n）=W(n)

于是就可以通过递推的方法，从后(F（n）)往前逐个找出所有的 F（i），再从中找一个最大的即为问题 2 的解。对于具体所走的路径（问题 1），可以通过一个向后的链接来实现，

具体请看下面的程序清单和注解。

**【参考程序】**

```
program Mine;
var f:array[1..200,1..2] of longint;
    a:array[1..200,1..200] of boolean;
    w:array[1..200] of longint;
    n,i,j,x,y,L,k,max:longint;
begin
  assign(input,'mine.in');
  assign(output,'mine.out');
  reset(input); rewrite(output);
  fillchar(a,sizeof(a),false);
  fillchar(f,sizeof(f),0);
  readln(n);
  for i:=1 to n do
  read(w[i]);readln;
  repeat
    readln(x,y);
    if (x<>0) and (y<>0) then a[x,y]:=true;
  until (x=0) and (y=0);
  f[n,1]:=w[n];
  for i:=n-1 downto 1 do
    begin
      L:=0;
      k:=0;
      for j:=i+1 to n do
      if a[i,j] and (f[j,1]>L) then
                begin  L:=f[j,1]; k:=j; end;
      f[i,1]:=L+w[i];
      f[i,2]:=k;
    end;
  L:=1;
  for j:=2 to n do
     if f[j,1]>f[L,1] then L:=j;
  max:=f[L,1];write(L);
  L:=f[L,2];
  while L<>0 do
    begin
      write('-',L);
      L:=f[L,2];
    end;
  writeln;
  writeln(max);
  close(input);close(output);
end.
```

## 例 2、防卫导弹（Catcher.pas）

**【问题描述】**

一种新型的防卫导弹可截击多个攻击导弹。它可以向前飞行，也可以用很快的速度向下飞行，可以毫无损伤地截击进攻导弹，但不可以向后或向上飞行。但有一个缺点，尽管它发射时可以达到任意高度，但它只能截击比它上次截击导弹时所处高度低或者高度相同的导弹。现对这种新型 防卫导弹进行测试，在每一次测试中，发射一系列的测试导弹（这些导弹发射的间隔时间固定，飞行速度相同），该防卫导弹所能获得的信息包括各进攻导弹的高度，以及它们发射次序。现要求编一程序，求在每次测试中，该防卫导弹最多能截击的进攻导弹数量，一个导弹能被截击应满足下列两个条件之一：

1、它是该次测试中第一个被防卫导弹截击的导弹；

2、它是在上一次被截击导弹的发射后发射，且高度不大于上一次被截击导弹的高度的导弹。

**【输入格式】**

从当前目录下的文本文件“CATCHER.IN”读入数据。该文件的第一行是一个整数 N（0〈=N〈=4000〉，表示本次测试中，发射的进攻导弹数，以下 N 行每行各有一个整数 hi（0〈=hi〈=32767〉，表示第 i 个进攻导弹的高度。文件中各行的行首、行末无多余空格，输入文件中给出的导弹是按发射顺序排列的。

**【输出格式】**

答案输出到当前目录下的文本文件“CATCHER.OUT”中，该文件第一行是一个整数 max，表示最多能截击的进攻导弹数，以下的 max 行每行各有一个整数，表示各个被截击的进攻导弹的编号（按被截击的先后顺序排列）。输出的答案可能不唯一，只要输出其中任一解即可。

**【输入输出样例】**

| 输入文件：CATCHER.IN |
|---|
| 3<br>25<br>36<br>23 |

| 输出文件：CATCHER.OUT |
|---|
| 2<br>1<br>3 |

题目讲得很麻烦,归根结底就是求一整串数中的最长不上升序列

这道题目一开始我使用回溯算法,大概可以拿到 1/3 的分吧,后来发现这其实是动态规划算法中最基础的题目,用一个二维数组 C[1..Max,1..2]来建立动态规划状态转移方程(注:C[1..Max,1]表示当前状态最多可击落的导弹数,C[1..Max,2]表示当前状态的前继标志):Ci=Max{C[j]+1,(j=i+1..n)},然后程序也就不难实现了.

**【参考程序】**

```
program catcher_hh;
var  f:text;
     i,j,k,max,n,num:integer;
     a:array [1..4000] of integer;          {导弹高度数组}
     c:array [1..4000,1..2] of integer;     {动态规划数组}
procedure readfile;
begin
   assign(f,'catcher.in'); reset(f);
   readln(f,num);
   for i:=1 to num do
   readln(f,a[i]);
end;
```

```
procedure work;
begin
   fillchar(c,sizeof(c),0); c[num,1]:=1;      {清空数组,赋初值}
   {开始进行动态规划}
   for i:=num-1 downto 1 do
   begin
       n:=0; max:=1;
       for j:=i+1 to num do
       if (a[i]>a[j]) and (max<1+c[j,1])
       then begin n:=j; max:=1+c[j,1]; end;
       c[i,1]:=max; c[i,2]:=n;
   end;
   writeln; writeln('Max : ',max);   {打印最大值}
   max:=0; n:=0;
   for i:=1 to num do
   if c[i,1]>max then begin max:=c[i,1]; n:=i; end;
   repeat  {返回寻找路径}
       writeln(n,' '); n:=c[n,2];
   until n=0;
end;
begin
   readfile; work;
end.
```

## 例 3、轮船问题(ship.pas)

**〖题目描述〗**

某国家被一条河划分为南北两部分，在南岸和北岸总共有 N 对城市，每一城市在对岸都有唯一的友好城市，任何两个城市都没有相同的友好城市。每一对友好城市都希望有一条航线来往，于是他们向政府提出了申请。由于河终年有雾。政府决定允许开通的航线就互不交叉（如果两条航线交叉，将有很大机会撞船）。

你的任务是编写一个程序来帮政府官员决定他们应拨款兴建哪些航线以使在安全条件下有最多航线可以被开通。

**【输入格式】**

输入文件(ship.in):包括了若干组数据，每组数据格式如下：

第一行两个由空格分隔的整数 x，y，10〈=x〈=6000，10〈=y〈=100。x 表示河的长度而 y 表示宽。第二行是一个整数 N(1<=N<=5000)，表示分布在河两岸的城市对数。接下来的 N 行每行有两个由空格分隔的正数 C，D（C、D〈=x〉，描述每一对友好城市沿着河岸与西边境线的距离，C 表示北岸城市的距离而 D 表示南岸城市的距离。在河的同一边，任何两个城市的位置都是不同的。整个输入文件以由空格分隔的两个 0 结束。

**【输出格式】**

输出文件(ship.out)：要在连续的若干行里给出每一组数据在安全条件下能够开通的最大航线数目。

**【输入输出样例】**

```
Ship.in            Ship.out
30  4              4
7
22  4
2   6
10  3
15  12
9   8
17  17
4   2
0   0
```

**【算法分析】**

对这道题的最一般想法是进行回溯，但是回溯对于数据规模达到 5000 的情况是不可行的。

于是我们改变一下思路，将每对友好城市看成一条线段，则这道题的描述化为：有 N 条线段，问最少去掉多少条线，可以使剩下的线段互不交叉？

顺理成章，删掉的线应该是和其它线交叉最多的，其"交叉数"最大。所谓"交叉数"是指某线与其它线的交叉情况，初始值为 0，若和其它线交叉则加 1 所得的值。按"交叉数"从大到小删除线，直到所有线都不交叉为止。此时，我们要解决的问题有：1、如何计算交叉数？2、怎么删线？

对第一个问题，以北岸为线的起点而南岸为线的终点；先将所有的线按照起点坐标值从小到大排序，按照每条线的终点坐标计算交叉数：求线 I 的交叉数 J[I]，则检查所有 1..I-1 条线，若线 J（ 1<=J<I ）的终点值大于线 I 的终点值，则线 I 与线 J 相交。J[I]与 J[J]同时加 1。整个搜索量最大为 5000 X 5000。

对第二个问题，将 J 数组从大到小排序，每删除一条线，则将与之相交的线的 J 值减 1，重复这个过程，直到所有 J 值都为 0。此时剩下的线则全不交叉。

如上数据，则可得下面结果：

| 编号 | 南岸 | 北岸 | 交叉数 |
|---|---|---|---|
| 1 | 1 | 3 | 1 |
| 2 | 2 | 4 | 2 |
| 3 | 3 | 1 | 2 |
| 4 | 4 | 5 | 1 |
| 5 | 4 | 2 | 2 |

此时，2、3、5 航线的交叉数都一样，如果删去的是 5、3 线，则剩下的 1、2、5 线互不相交，最多航线数为 3；但如果删去的是 2、3，则还要删去第 5 线才符合要求，此时的最多航线数为 2，不是最优解。

于是，我们从上面的分析中再深入，将航线按起点坐标排好序后，如上所述，在本题中，只要线 J 的起点小于线 I 的起点，同时它的终点也小于线 I 的终点，则线 J 和线 I 不相交。因此，求所有线中最多能有多少条线不相交，实际上是从终点坐标值数列中求一个最长不下降序列。这就把题目转化为一个非常典型的动态规划题目了。

求最长不下降序列的规划方程如下：

L(Si)=max{L(Sj)}+1；1< = j < i 且 Sj < Si。 Si 为航线的终点坐标值。

如上数据可以得下解：

| 编号 | 南岸 | 北岸 | L 值和前趋 |
|---|---|---|---|
| 1 | 1 | 3 | 1，0 |
| 2 | 2 | 4 | 2，1 |
| 3 | 3 | 1 | 1，0 |
| 4 | 4 | 5 | 3，2 |
| 5 | 4 | 2 | 2，3 |

非常明显，可以得出解为 3，航线为 4，2，1。

从以上分析过程可以得出：当我们拿到一道题时，不要急于求解，而应先将题目的表面现象一层层象剥竹笋一样去掉，只留下最实质的内容。这时再来设计算法，往往能事半功倍。

## 例 4、车队过桥(bridge.pas)

**〖题目描述〗**

GDOI 工作组遇到了一个运输货物的问题。现在有 N 辆车要按顺序通过一个单向的小桥，由于小桥太窄，不能有两辆车并排通过，所以在桥上不能超车。另外，由于小桥建造的时间已经很久，所以小桥只能承受有限的重量，记为 Max（吨）。所以，车辆在过桥的时候必须要有管理员控制，将这 N 辆车按初始顺序分组，每次让一个组过桥，并且只有在一个组中所有的车辆全部过桥以后才让下一组车辆上桥。现在，每辆车的重量和最大速度是已知的，而每组车的过桥时间由该组中速度最慢的那辆车决定。现在请你编一个程序，将这 N 辆车分组，使得全部车辆通过小桥的时间最短。

**【输入格式】**

数据存放在当前目录下的文本文件“bridge.in”中。
文件的第一行有三个数，分别为 Max（吨），Len（桥的长度，单位：Km），N（三个数之间用一个或多个空格分开）。
接下来有 N 行，每行两个数，第 i 行的两个数分别表示第 i 辆车的重量(吨)和最大速度(m/s)。
注意：所有的输入都为整数，并且任何一辆车的重量都不会超过 Max。

**【输出格式】**

答案输出到当前目录下的文本文件“bridge.out”中。
文件只有一行，输出全部车辆通过小桥的最短时间（s），精确到小数点后一位。

**【输入输出样例】**

bridge.in　　bridge.out

```
100 5  9
40 25
50 20
70 10
12 50
9 70
49 30
38 25
27 50
19 70
```

```
1050.0
```

**【参考程序】**

```
program bridge;
  var
      a:array [0..1000] of real;
      b:array [1..1000,1..2] of longint;
      max,len,n:longint;
      c:longint;
     f1,f2:Text;
  procedure try;
    var
      c,d,e,f,g:longint; flag:boolean;
      time,t,t1:real;
    begin
      a[0]:=0; a[1]:=(len/b[1,2]);
      for c:=2 to n do begin
        flag:=true; d:=c+1; e:=0; f:=maxint; time:=maxint;
        while flag do begin
          d:=d-1;
          if (e+b[d,1])<=max then begin
                                  e:=e+b[d,1]; if b[d,2]<f then f:=b[d,2]; end
                            else flag:=false;
          if flag then begin
            t1:=len/f; t:=t1+a[d-1];
            if t<time then time:=t;
            if d=1 then flag:=false;
          end;
        end;
        a[c]:=time;
     end;
     rewrite (f2); writeln (f2,a[n]*60:1:1);
     close (f2);
   end;

  BEGIN
    assign (f1,'bridge.in');
    assign (f2,'bridge.out');
    reset (f1);
    readln (f1,max,len,n);
    for c:=1 to n do
      readln (f1,b[c,1],b[c,2]);
    close (f1);
    try;
  END.
```

## 例 5、复制书稿（book.pas）

**【问题描述】**

有 M 本书（编号为 1，2，…，M），每本书都有一个页数（分别是 $P_1$，$P_2$，…，$P_M$）。想将每本都复制一份。将这 M 本书分给 K 个抄写员（1<=K<=M<=500），每本书只能分配给一个抄写员进行复制。每个抄写员至少被分配到一本书，而且被分配到的书必须是连续顺序的。复制工作是同时开始进行的，并且每个抄写员复制一页书的速度都是一样的。所以，复制完所有书稿所需时间取决于分配得到最多工作的那个抄写员的复制时间。试找一个最优分配方案，使分配给每一个抄写员的页数的最大值尽可能小。

**【输入格式】**

第一行两个整数 M、K；（K<=M<=100）

第二行 M 个整数，第 i 个整数表示第 i 本书的页数。

**【输出格式】**

共 K 行，每行两个正整数，第 i 行表示第 i 个人抄写的书的起始编号和终止编号。K 行的起始编号应该从小到大排列，如果有多解，则尽可能让前面的人少抄写。

**【输入样例】**BOOK.IN

```
9 3
1 2 3 4 5 6 7 8 9
```

**【输出样例】**BOOK.OUT

```
1 5
6 7
8 9
```

**【算法分析】**

该题中 M 本书是顺序排列的，K 个抄写员选择数也是顺序且连续的。不管以书的编号，还是以抄写员标号作为参变量划分阶段，都符合策略的最优化原理和无后效性。考虑到 K<=M，以抄写员编号来划分阶段会方便些。

设 F（I，J）为前 I 个抄写员复制前 J 本书的最小“页数最大数”。于是便有 F（I，J）=MIN{F（I-1，V），T（V+1，J）} （1<=I<=K，I<=J<=M-K+I，I-1<=V<=J-1）。 其中 T（V+1，J）表示从第 V+1 本书到第 J 本书的页数和。边界条件 F（1,i）=T(1,j)。

**【参考程序】**

```
program book;
const maxm=100;
var p,t:array[1..maxm] of longint;
    f:array[1..maxm,1..maxm] of longint;
    m,k,i,j,v:longint;

function max(a,b:longint):longint;
begin
     if a>b then max:=a else max:=b;
end;
```

```
procedure out(i,j:longint);    {递归输出}
var v:longint;
begin
     if i=1 then
        begin
             writeln(1,' ',j);
             exit;
        end;
     for v:=i-1 to j-1 do
         if max(f[i-1,v],t[j]-t[v])<=f[k,m] then
            begin
                 out(i-1,v);
                 writeln(v+1,' ',j);
                 exit;
            end;
end;
BEGIN  {main}
     assign(input,'book.in');
     assign(output,'book.out');
     reset(input);rewrite(output);
     read(m,k);
     for i:=1 to m do read(p[i]);        {p[i]存放每本书的页数}
     t[1]:=p[1];
     for i:=2 to m do t[i]:=t[i-1]+p[i];  {t[i]存放前 i 本书的总页数}
     for j:=1 to m do f[1,j]:=t[j];       {边界条件、初始状态}
     for i:=2 to k do
         for j:=i to m do                 {记忆化递归}
             begin
                  f[i,j]:=maxlongint;
                  for v:=i-1 to j-1 do
                      if f[i-1,v]>t[j]-t[v]
                         then begin
                                   if f[i-1,v]<f[i,j] then
                                      f[i,j]:=f[i-1,v];
                              end
                         else begin
                                   if t[j]-t[v]<f[i,j] then
                                      f[i,j]:=t[j]-t[v];
                              end;
             end;
     out(k,m);
     close(input); close(output);
END.
```

## 例 6、拔河比赛（tug.pas）

**【问题描述】**

一个学校举行拔河比赛，所有的人被分成了两组，每个人必须（且只能够）在其中的一组，要求两个组的人数相差不能超过 1，且两个组内的所有人体重加起来尽可能地接近。

**【输入格式】**

输入数据的第 1 行是一个 n，表示参加拔河比赛的总人数，n<=100，接下来的 n 行表示第 1 到第 n 个人的体重，每个人的体重都是整数（1<=weight<=450）。

**【输出格式】**

输出数据应该包含两个整数：分别是两个组的所有人的体重和，用一个空格隔开。注意如果这两个数不相等，则请把小的放在前面输出。

**【输入样例】**tug.in

```
3
100
90
200
```

**【输出样例】**tug.out

```
190 200
```

**【算法分析】**

这道题目不满足动态规划最优子结构的特性。因为最优子结构要求一个问题的最优解只取决于其子问题的最优解。就这道题目而言，当前 n-1 个人的分组方案达到最优时，并不意味着前 n 个人的分组方案也最优。但题目中标注出每个人的最大体重为 450，这就提醒我们可以从这里做文章，否则的话，命题者大可把最大体重标注到长整型。假设 w[i]表示第 i 个人的体重。c[i,j,k]表示在前 i 个人中选 j 个人在一组，他们的重量之和等于 k 是否可能。显然，c[i,j,k]是 boolean 型，其值为 true 代表有可能，false 代表没有可能。那 c[i,j,k]与什么有关呢？从前 i 个人中选出 j 个人的方案，不外乎两种情况：⑴第 i 个人没有被选中，此时就和从前面 i-1 个人中选出 j 个人的方案没区别，所以 c[i,j,k]与 c[i-1,j,k]有关。⑵第 i 个人被选中，则 c[i,j,k]与 c[i-1,j-1,k-w[i]]有关。综上所述，可以得出：

c[i,j,k]=c[i-1,j,k] or c[i-1,j-1,k-w[i]]。

这道题占用的空间看似达到三维，但因为 i 只与 i-1 有关，所以在具体实现的时候，可以把第一维省略掉。另外在操作的时候，要注意控制 j 与 k 的范围(0<=j<=i/2,0<=k<=j*450)，否则有可能超时。

这种方法的实质是把解本身当作状态的一个参量，把最优解问题转化为判定性问题，用递推的方法求解。这种问题有一个比较明显的特征，就是问题的解被限定在一个较小的范围内，如这题中人的重量不超过 450。

**【参考程序】**

```
program Tug;
const  maxn=100;
       maxn2=maxn div 2;
       maxrange=450;
var  c:array [0..maxn2,0..maxn2*maxrange] of boolean;
```

```
    w:array [1..maxn] of longint;
    n,n2,sum,i,j,k,min,ans:longint;
BEGIN
  assign(input,'tug.in');
  assign(output,'tug.out');
  reset(input);
  rewrite(output);
  read(n); n2:=(n+1) div 2;
  sum:=0;
  for i:=1 to n do
    begin
      read(w[i]);
      inc(sum,w[i]);
    end;
  fillchar(c,sizeof(c),0);
  c[0,0]:=true;
  for i:=1 to n do
      for j:=n2-1 downto 0 do
        for k:=maxrange*i downto 0 do
          if c[j,k] then c[j+1,k+w[i]]:=true;
  min:=maxlongint;
  ans:=0;
  for k:=0 to maxrange*n2 do
      if c[n2,k] and (abs(sum-k-k)<min) then
      begin
        min:=abs(sum-k-k);
        if k<=sum div 2 then ans:=k else ans:=sum-k;
      end;
  write(ans,' ',sum-ans);
  close(output); close(input);
END.
```

## 例 7、投资问题（invest.pas）

**【问题描述】**

有 n 万元的资金，可投资于 m 个项目，其中 m 和 n 为小于 100 的自然数。对第 i（1≤i≤m）个项目投资 j 万元(1≤j≤n，且 j 为整数)可获得的回报为 Q（i,j)，请你编一个程序，求解并输出最佳的投资方案（即获得回报总值最高的投资方案)。

输入数据放在文件 invest.in 中，格式如下：

```
m  n
Q(1,0)    Q(1,1)……Q(1,n)
Q(2,0)    Q(2,1)……Q(2,n)
……
Q(m,0)    Q(m,1)……Q(m,n)
```

输出数据放在文件 invest.out 中，格式为：

r(1)　r(2)　••••••　r(m)　P

其中 r(i)（1≤i≤m）表示对第 i 个项目的投资万元数，P 为总的投资回报值，保留两位有效数字，任意两个数之间空一格。当存在多个并列的最佳投资方案时，只要求输出其中之一即可。

**【输入样例】**

invest.in：

```
2  3
0  1.1  1.3  1.9
0  2.1  2.5  2.6
```

**【输出样例】**

invest.out：

```
1  2  3.6
```

**【算法分析】**

本题可供考虑的递推角度只有资金和项目两个角度，从项目的角度出发，逐个项目地增加可以看出只要算出了对前 k 个项目投资 j 万元最大投资回报值(1≤j≤n)，就能推出增加第 k+1 个项目后对前 k+1 个项目投资 j 万元最大投资回报值(1≤j≤n)，设 P[k,j]为前 k 个项目投资 j 万元最大投资回报值，则 P[k+1,j]= Max(P[k,i]+Q[k+1,j-i]),0<=i<=j，k=1 时，对第一个项目投资 j 万元最大投资回报值(0≤j≤n)是已知的（边界条件）。

**【算法设计】**

动态规划的时间复杂度相对于搜索算法是大大降低了，却使空间消耗增加了很多。所以在设计动态规划的时候，需要尽可能节省空间的占用。本题中如果用二维数组存储原始数据和最大投资回报值的话，将造成空间不够，事实上，从前面的问题分析可知：在计算对前 k+1 个项目投资 j 万元最大投资回报值时，只要用到矩阵 Q 的第 k+1 行数据和对前 k 个项目投资 j 万元最大投资回报值，而与前面的数据无关，后者也只需有一个一维数组存储即可，程序中用一维数组 Q 存储当前项目的投资回报值，用一维数组 maxreturn 存储对当前项目之前的所有项目投资 j 万元最大投资回报值(0≤j≤n)，用一维数组 temp 存储对到当前项目为止的所有项目投资 j 万元最大投资回报值(0≤j≤n)。为了输出投资方案，程序中使用了一个二维数组 invest，invest[k,j]记录了对前 k 个项目投资 j 万元获得最大投资回报时投资在第 k 个项目上的资金数。

**【参考程序】**

```
program invest(input,output);
const maxm=100; maxn=100;
type arraytype=array [0..maxn] of real;
var i,j,k,m,n,rest:integer;
    q,maxreturn,temp:arraytype;
    invest:array[1..maxm,0..maxn] of integer;
    result:array[1..maxm] of integer;
begin
     assign(input,' invest.in' ); reset(input);
     readln(m,n);
```

```
    for j:=0 to n do read(q[j]);
    readln;
    for i:=1 to m do
        for j:=0 to n do invest[i,j]:=0;
    maxreturn:=q;
    for j:=0 to n do invest[1,j]:=j;
    for k:=2 to m do
    begin
         temp:=maxreturn;
         for j:=0 to n do invest[k,j]:=0;
         for j:=0 to n do read(q[j]);
         readln;
         for j:=0 to n do
             for i:=0 to j do
                 if maxreturn[j-i]+q[i]>temp[j] then
                    begin
                         temp[j]:=maxreturn[j-i]+q[i];
                         invest[k,j]:=i
                    end;
         maxreturn:=temp
    end;
    close(input);
    assign(output,' invest.out' );rewrite(output);
    rest:=n;
    for i:=m downto 1 do
    begin
         result[i]:=invest[i,rest];
         rest:=rest-result[i]
    end;
    for i:=1 to m do write(result[i],' ');
    writeln(maxreturn[n]:0:2);
    close(output)
end.
```

## 例 8、花店橱窗布置问题(FLOWER.PAS)

**【问题描述】**

假设你想以最美观的方式布置花店的橱窗。现在你有 F 束不同品种的花束，同时你也有至少同样数量的花瓶被按顺序摆成一行。这些花瓶的位置固定于架子上，并从 1 至 V 顺序编号，V 是花瓶的数目，从左至右排列，则最左边的是花瓶 1，最右边的是花瓶 V。花束可以移动，并且每束花用 1 至 F 间的整数唯一标识。标识花束的整数决定了花束在花瓶中的顺序，如果 I＜J，则令花束 I 必须放在花束 J 左边的花瓶中。

例如，假设一束杜鹃花的标识数为 1，一束秋海棠的标识数为 2，一束康乃馨的标识数为 3，所有的花束在放入花瓶时必须保持其标识数的顺序，即：杜鹃花必须放在秋海棠左边的花瓶中，秋海棠必须放在康乃馨左边的花瓶中。如果花瓶的数目大于花束的数目。则多余的花瓶必须空置，且每个花瓶中只能放一束花。

每一个花瓶都具有各自的特点。因此，当各个花瓶中放入不同的花束时，会产生不同的美学效果，并以美学值（一个整数）来表示，空置花瓶的美学值为零。

在上述例子中，花瓶与花束的不同搭配所具有的美学值，如下表所示。

| | | 花瓶 | | | | |
|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 |
| 花束 | 1（杜鹃花） | 7 | 23 | -5 | -24 | 16 |
| | 2（秋海棠） | 5 | 21 | -4 | 10 | 23 |
| | 3（康乃馨） | -21 | 5 | -4 | -20 | 20 |

例如，根据上表，杜鹃花放在花瓶 2 中，会显得非常好看；但若放在花瓶 4 中则显得十分难看。

为取得最佳美学效果，你必须在保持花束顺序的前提下，使花束的摆放取得最大的美学值。如果有不止一种的摆放方式具有最大的美学值，则其中任何一直摆放方式都可以接受，但你只要输出任意一种摆放方式。

**1、假设条件**

$1 \leqslant F \leqslant 100$，其中 F 为花束的数量，花束编号从 1 至 F。

$F \leqslant V \leqslant 100$，其中 V 是花瓶的数量。

$-50 \leqslant A_{ij} \leqslant 50$，其中 $A_{ij}$ 是花束 i 在花瓶 j 中的美学值。

**2、输入**

输入文件是 flower.in。

第一行包含两个数：F，V。

随后的 F 行中，每行包含 V 个整数，$A_{ij}$ 即为输入文件中第（i+1 ）行中的第 j 个数。

**3、输出**

输出文件必须是名为 flower.out 的正文文件，文件应包含两行：

第一行是程序所产生摆放方式的美学值。

第二行必须用 F 个数表示摆放方式，即该行的第 K 个数表示花束 K 所在的花瓶的编号。

**4、例子**

flower.in:

```
3 5
7 23 -5 -24 16
5 21 -4 10 23
-21 5 -4 -20 20
```

flower.out:

```
53
2 4 5
```

**【算法分析】**

flower 一题是 IOI99 第一天第一题，该题如用组合的方法处理，将会造成超时。正确的方法是用动态规划，考虑角度为一束一束地增加花束，假设用 b(i,j)表示 1～i 束花放在 1 到 j 之间的花瓶中的最大美学值，其中 i<=j ，则 b(i,j)=max(b[i-1,k-1]+A[i,k])，其中 i<=k<=j，A(i,k)的含义参见题目。输出结果时，显然使得 b[F,k]取得总的最大美观值的第一个 k 值就是第 F 束花应该摆放的花瓶位置，将总的最大美观值减去 A[i,k]的值即得到前 k-1 束花放在前 k-1 个瓶中的最大美观值，依次使用同样的方法就可求出每一束花应该摆放的花瓶号。由于这一过程是倒推出来的，所以程序中用递归程序来实现。

**【参考程序】**

```
program ex8_4;
const max=100;
var f,v,i,j,k,cmax,current,max_val:integer;
    table,val:array[1..max,1..max] of integer;

procedure print(current,max_val:integer);
var i:integer;
begin
     if current>0 then
     begin
          i:=current;
          while val[current,i]<>max_val do i:=i+1;
          print(current-1,max_val-table[current,i]);
          write(i,' ')
     end
end;
begin
     assign(input,'flower.in');
     assign(output,'flower.out');
     reset(input);rewrite(output);
     readln(f,v);
     for i:=1 to f do
     begin
          for j:=1 to v do read(table[i,j]);
          readln;
     end;
     close(input);
     max_val:=-maxint;
     for i:=1 to v do
         if max_val<table[1,i]
            then begin val[1,i]:=table[1,i];max_val:=table[1,i] end
            else val[1,i]:=table[1,i];
     for i:=2 to f do
         for j:=i to v-f+i do
         begin
```

```
            max_val:=-maxint;
            for k:=i-1 to j-1 do
            begin
                cmax:=-maxint;
                for current:=k+1 to j do
                    if table[i,current]>cmax then cmax:=table[i,current];
                if cmax+val[i-1,k]>max_val then max_val:=cmax+val[i-1,k]
            end;
            val[i,j]:=max_val
        end;
    max_val:=-maxint;
    for i:=f to v do
        if val[f,i]>max_val then max_val:=val[f,i];
    writeln(max_val);
    print(f,max_val);
    writeln
end.
```

## 例 9、方格取数（pane.pas）

**【问题描述】**

设有 N*N 的方格图（N≤8），我们将其中的某些方格中填入正整数，而其他的方格中则放入数字 0。如下图所示（见样例）：

向右

向下

| A | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 13 | 0 | 0 | 6 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 7 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 14 | 0 | 0 | 0 | 0 |
| 5 | 0 | 21 | 0 | 0 | 0 | 4 | 0 | 0 |
| 6 | 0 | 0 | 15 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 14 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

B

某人从图的左上角的 A 点出发，可以向下行走，也可以向右走，直到到达右下角的 B 点。在走过的路上，他可以取走方格中的数（取走后的方格中将变为数字 0）。

此人从 A 点到 B 点共走两次，试找出 2 条这样的路径，使得取得的数之和为最大。

**【输入格式】**

输入文件 pane.in，第一行为一个整数 N（表示 N*N 的方格图），接下来的每行有三个整数，前两个表示位置，第三个数为该位置上所放的数。一行单独的 0 表示输入结束。

**【输出格式】**

输出文件 pane.out，只需输出一个整数，表示 2 条路径上取得的最大的和。

**【输入样例】**

```
8
2  3  13
2  6  6
3  5  7
4  4  14
5  2  21
5  6  4
6  3  15
7  2  14
0  0  0
```

**【输出样例】**

```
67
```

**【算法分析】**

本题是从 1997 年国际信息学奥林匹克的障碍物探测器一题简化而来，如果人只能从 A 点到 B 点走一次，则可以用动态规划算法求出从 A 点到 B 点的最优路径。具体的算法描述如下：从 A 点开始，向右和向下递推，依次求出从 A 点出发到达当前位置（i，j）所能取得的最大的数之和，存放在 sum 数组中，原始数据经过转换后用二维数组 data 存储，为方便处理，对数组 sum 和 data 加进零行与零列，并置它们的零行与零列元素为 0。易知

$$sum[i,j]=\begin{cases} data[i,j] & \text{当 } i=0 \text{ 或 } j=0 \\ \max(sum[i-1,j],sum[i,j-1])+data[i,j] & \text{当 } i>0,\text{且 } j>0 \end{cases}$$

求出 sum 数组以后，通过倒推即可求得最优路径，具体算法如下：

```
置 sum 数组零行与零列元素为 0
for i:=1 to n do
    for j:=1 to n do
        if sum[i-1,j]>sum[i,j-1]
          then sum[i,j]:=sum[i-1,j]+data[i,j]
          else sum[i,j]:=sum[i,j-1]+data[i,j];
 i:=n; j:=n;
 while (i>1) or (j>1) do
       if (i>1) and (sum[i,j]=sum[i-1,j]+data[i,j])
          then begin data[i,j]:=-1; i:=i-1 end
          else begin data[i,j]:=-1; j:=j-1 end;
 data[1,1]:=-1;
```

凡是被标上-1 的格子即构成了从 A 点到 B 点的一条最优路径。

那么是否可以按最优路径连续走两次而取得最大数和呢?这是一种很自然的想法,并且对样例而言也是正确的,具体算法如下：

求出数组 sum，

s1:=sum[n,n]，

求出最优路径，

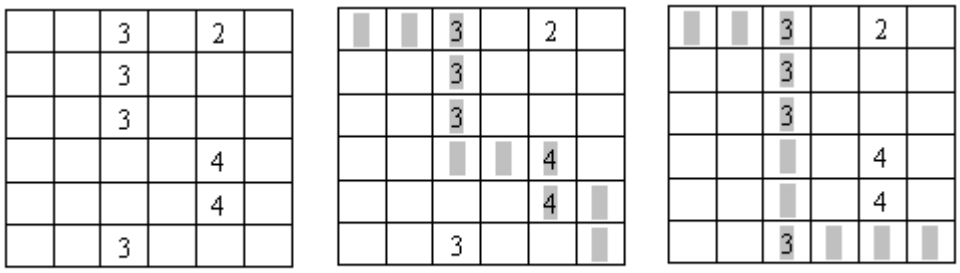
将最优路径上的格子中的值置为 0，

将数组 sum 的所有元素置为 0，

第二次求出数组 sum，

输出 s1+sum[n,n]。

虽然这一算法保证了连续的两次走法都是最优的，但却不能保证总体最优，相应的反例也不难给出，请看下图：



图一　　图二　　图三

图二按最优路径走一次后，余下的两个数 2 和 3 就不可能同时取倒了，而按图三中的非最优路径走一次后却能取得全部的数，这是因为两次走法之间的协作是十分重要的，而图 2 中的走法并不能考虑全局，因此这一算法只能算是“贪心算法”。虽然在一些情况下，贪心算法也能够产生最优解，但总的来说“贪心算法”是一种有明显缺陷的算法。

既然简单的动态规划行不通，那么看看穷举行不行呢？因为问题的规模比较小，启发我们从穷举的角度出发去思考，首先让我们来看看 N=8 时，从左上角 A 到达右下角 B 的走法共有多少种呢？显然从 A 点到 B 点共需走 14 步，其中向右走 7 步，向下走 7 步，共有 $C_{14}^{7}$=3432 种不同的路径，走两次的路径组合总数为 $C_{3432}^{2}$=3432＊3431/2=5887596，从时间上看是完全可以承受的，但是如果简单穷举而不加优化的话，对极限数据还是会超时的，优化的最基本的方法是以空间换时间，具体到本题就是预先将每一条路径以及路径上的数字之和（称之为路径的权 weight）求出并记录下来，然后用双重循环穷举任意两条不同路径之组合即可。

考虑到记录所有的路径需要大量的存储空间，我们可以将所有的格子逐行进行编号，这样原来用二维坐标表示的格子就变成用一个 1 到 $n^2$ 之间的自然数来表示，格子（i,j）对应的编号为（i-1）*n+j。一条路径及其权使用以下的数据结构表示：

```
const maxn=8;
type arraytype=array[1..2*maxn-2] of byte;
    recordtype=record path:arraytype;
                      weight:longint
                end;
```

数组 path 依次记录一条路径上除左上和右下的全部格子的编号。

将所有的路径以及路径的权求出并记录下来的算法可用一个递归的过程实现，其中 i,j 表示当前位置的行与列，step 表示步数，sum 记录当前路径上到当前位置为止的数之和，当前路径记录在数组 position 中（不记录起始格），主程序通过调用 try(1,1,0,data[1,1]) 求得所有路径。

```
procedure try(i,j,step,sum:longint);
begin
     if (i=n) and (j=n)
        then begin
                  total:=total+1;
                  a[total].path:=position;
                  a[total].weight:=sum;
             end
```

```
        else begin
                  if i+1<=n then
                  begin
                       position[step]:=i*n+j;
                       try(i+1,j,step+1,sum+data[i+1,j])
                  end;
                  if j+1<=n then
                  begin
                       position[step]:=(i-1)*n+j+1;
                       try(i,j+1,step+1,sum+data[i,j+1])
                  end
            end
end;
```

在穷举了二条不同的路径后，只要将二条路径的权相加再减去二条路径中重叠格子中的数即为从这二条路径连走两次后取得的数之和，具体算法如下：

```
 for i:=1 to n do  {将二维数组转化为一维数组}
     for j:=1 to n do d1[(i-1)*n+j]:=data[i,j];
 max:=0;
 for i:=1 to total-1 do
     for j:=i+1 to total do
     begin
          current:=a[i].weight+a[j].weight;
          for k:=1 to 2*n-3 do {判断重叠格子，但不包括起点的终点}
          begin
               if a[i].path[k]=a[j].path[k]{第 k 步到达同一方格}
                  then current:=current-d1[a[i].path[k]]
          end;
          if current>max then max:=current
     end;
 writeln(max-data[1,1]-data[n,n]);
```

应该看到穷举的效率是十分低下的，如果问题的规模再大一些，穷举法就会超时，考虑到走一次可以使用动态规划，则只需穷举走第一次的路径，而走第二次可以用动态规划，这样可大大提高程序的效率，其算法复杂度为$o(n^2C_{2n-2}^{n-1})$，实现时只需将前面二种算法结合起来即可，但这样做仍然不能使人满意，因为只要穷举了从 A 点到 B 点所有路径，算法的效率就不可能很高，要想对付尽可能大的 n，还是要依靠动态规划算法。实际上本问题完全可以用动态规划解决，只是递推起来更为复杂些而已，前面在考虑只走一次的情况，只需考虑一个人到达某个格子（i，j）的情况，得出 sum[i，j]=max（sum[i-1，j]，sum[i，j-1]）+data[i，j]，现在考虑两个人同时从 A 出发，则需考虑两个人到达任意两个格子(i1，j1)与(i2，j2)的情况，显然要到达这两个格子，其前一状态必为(i1-1，j1)，(i2-1，j2)；(i1-1，j1)，(i2，j2-1)；(i1，j1-1)，(i2-1，j2)；(i1，j1-1)，(i2，j2-1) 四种情况之一，类似地，可以推导出：

设
p=max(sum[i1-1,j1,i2-1,j2],sum[i1-1,j1,i2,j2-1],sum[i1,j1-1,i2-1,j2],sum[i1,j1-1,i2,j2-1])，则

$$sum[i1,j1,i2,j2]=\begin{cases} 0 & \text{当 } i1=0 \text{ 或 } j1=0 \text{ 或 } i2=0 \text{ 或 } j2=0 \\ p + data[i1,j1] & \text{当 } i1,j1,i2,j2 \text{ 均不为零，且 } i1=i2,\ j1=j2 \\ p + data[i1,j1]+data[i2,j2] & \text{当 } i1,j1,i2,j2 \text{ 均不为零,且 } i1 \neq i2 \text{ 或 } j1 \neq j2 \end{cases}$$

具体算法如下：

```
     置 sum 数组所有元素全为 0;
     for i1:=1 to n do
         for j1:=1 to n do
             for i2:=1 to n do
                 for j2:=1 to n do
                 begin
                      if sum[i1-1,j1,i2-1,j2]>sum[i1,j1,i2,j2]
                         then sum[i1,j1,i2,j2]:=sum[i1-1,j1,i2-1,j2];
                      if sum[i1-1,j1,i2,j2-1]>sum[i1,j1,i2,j2]
                         then sum[i1,j1,i2,j2]:=sum[i1-1,j1,i2,j2-1];
                      if sum[i1,j1-1,i2-1,j2]>sum[i1,j1,i2,j2]
                         then sum[i1,j1,i2,j2]:=sum[i1,j1-1,i2-1,j2];
                      if sum[i1,j1-1,i2,j2-1]>sum[i1,j1,i2,j2]
                         then sum[i1,j1,i2,j2]:=sum[i1,j1-1,i2,j2-1];
                      sum[i1,j1,i2,j2]:=sum[i1,j1,i2,j2]+data[i1,j1];
                      if (i1<>i2) or (j1<>j2)
                         then sum[i1,j1,i2,j2]:=sum[i1,j1,i2,j2]+data[i2,j2]
                 end;
    writeln(sum[n,n,n,n])
```

**【参考程序】**

```
program pane;
const maxn=10;
type arraytype=array [0..maxn,0..maxn] of longint;
var i,j,k,n,i1,i2,j1,j2:longint;
    data:arraytype;
   sum:array [0..maxn,0..maxn,0..maxn,0..maxn] of longint;

function max(x,y:longint):longint;
begin
     if x>y then max:=x else max:=y;
end;

BEGIN  {main}
     Assign(input,' pane.in' );
     Assign(output,' pane.out' );
     Reset(input);Rewrite(output);
```

```
    for i:=1 to maxn do
        for j:=1 to maxn do data[i,j]:=0;
    readln(n);
    repeat
          readln(i,j,k);
          data[i,j]:=k
    until (i=0) and (j=0) and (k=0);
    fillchar(sum,sizeof(sum),0);
    for i1:=1 to n do
        for j1:=1 to n do
            for i2:=1 to n do
                for j2:=1 to n do
                begin
                     if sum[i1-1,j1,i2-1,j2]>sum[i1,j1,i2,j2]
                        then sum[i1,j1,i2,j2]:=sum[i1-1,j1,i2-1,j2];
                     if sum[i1-1,j1,i2,j2-1]>sum[i1,j1,i2,j2]
                        then sum[i1,j1,i2,j2]:=sum[i1-1,j1,i2,j2-1];
                     if sum[i1,j1-1,i2-1,j2]>sum[i1,j1,i2,j2]
                        then sum[i1,j1,i2,j2]:=sum[i1,j1-1,i2-1,j2];
                     if sum[i1,j1-1,i2,j2-1]>sum[i1,j1,i2,j2]
                        then sum[i1,j1,i2,j2]:=sum[i1,j1-1,i2,j2-1];
                     sum[i1,j1,i2,j2]:=sum[i1,j1,i2,j2]+data[i1,j1];
                     if (i1<>i2) or (j1<>j2)
                        then sum[i1,j1,i2,j2]:=sum[i1,j1,i2,j2]+data[i2,j2]
                end;
    writeln(sum[n,n,n,n]);
    close(input);close(output)
END.
```

**【运行示例】**

```
pane.in:
3
1 1 10
1 3 5
2 2 6
2 3 4
3 1 8
3 2 2
0 0 0

pane.out:
30
```

## 【上机练习】

例 1、挖地雷(Mine.pas)
例 2、防卫导弹(Catcher.pas)
例 3、轮船问题(ship.pas)
例 4、车队过桥(bridge.pas)
例 5、复制书稿(book.pas)
例 6、拔河比赛(tug.pas)
例 7、投资问题(invest.pas)
8、橱窗布置（Flower.pas）
9、方格取数(pane.pas)
10、合并石子(Unite.pas)

**【问题描述】**

在一个操场上一排地摆放着N堆石子。现要将石子有次序地合并成一堆。规定每次只能选相邻的 2 堆石子合并成新的一堆，并将新的一堆石子数记为该次合并的得分。

**【问题求解】**

试设计一个程序，计算出将N堆石子合并成一堆的最小得分。

**【输入格式】**

第一行为一个正整数N (2≤N≤100)；

以下N行,每行一个正整数，小于 10000，分别表示第 i 堆石子的个数(1≤i≤N)。

**【输出格式】**

为一个正整数，即最小得分。

**【输入样例】**

UNITE.IN
7
13
7
8
16
21
4
18

**【输出样例】**

UNITE.OUT
239

# 第十五章　递推关系在竞赛中的应用

## 一、 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 $H_0, H_1, \cdots, H_n, \cdots$ 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 $H_i (0 \leq i < n)$ 联系起来，这样的式子就叫做递推关系。**

## 二、 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。

要求把 a 柱上 n 个圆盘按下述规则移到 c 柱上：

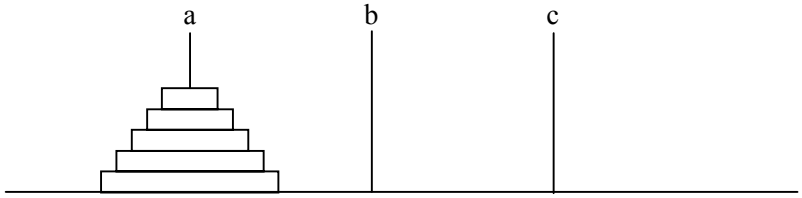



图 1

(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≥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。

$\therefore h_n=2h_{n-1}+1$　　边界条件：$h_1=1$

## *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。

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：$a_2-a_1=2$；

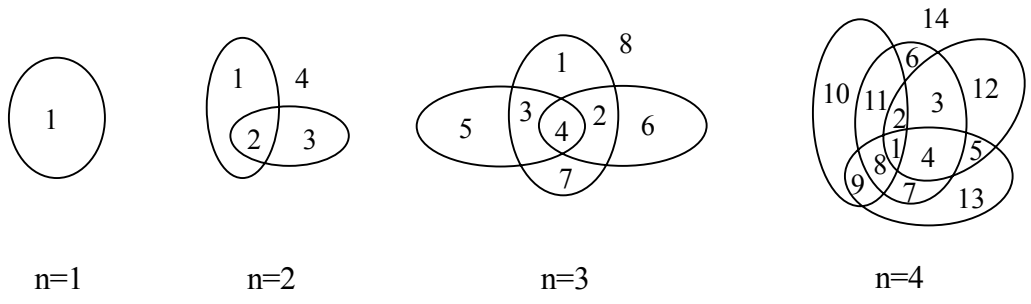



图 2

$a_3-a_2=4$；$a_4-a_3=6$。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 $h_n$。

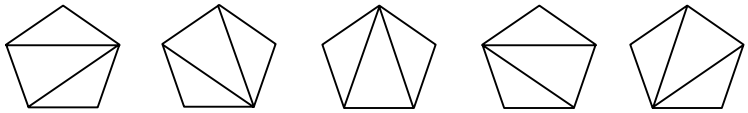


图 3

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△$P_1P_kP_n$ 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 $\sum_{i=2}^{n-1} C_i C_{n-i+1}$，同时考虑到计算的方便，约定边界条件 $C_2=1$。

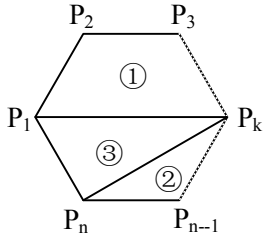



图 4

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 $b_1, b_2, \cdots\cdots b_n$ 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 $b_1, b_2, \cdots\cdots b_{n-1}$ 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。

## 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 $F_3, F_4 \cdots\cdots F_n$，而利用母函数的方法，则可以得出公式 $F_n=\dfrac{\left(\frac{1+\sqrt{5}}{2}\right)^n-\left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$，则可直接求出所需要的任意一项。

# 三、 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。

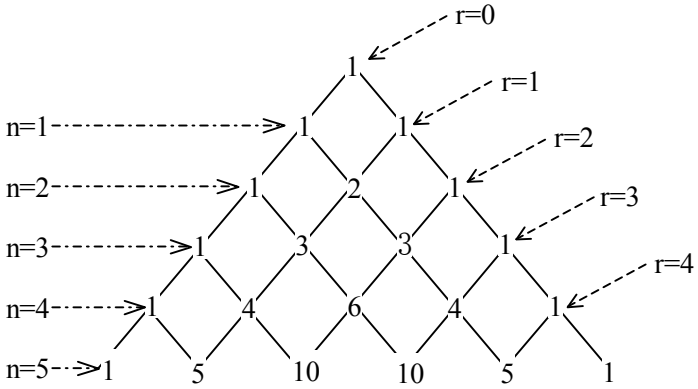



图 5

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。**

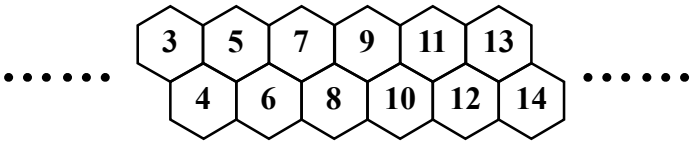



图 6

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 $f_n=f_{n-1}+f_{n-2}$ $(a+2\le n\le b)$，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≤500)条直线，已知有 p(p≥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≥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：**

| | | | | | | |
|---|---|---|---|---|---|---|
| 16 | 4 | 3 | 12 | 6 | 0 | 3 |
| 4 | -5 | 6 | 7 | 0 | 0 | 2 |
| 6 | 0 | -1 | -2 | 3 | 6 | 8 |
| 5 | 3 | 4 | 0 | 0 | -2 | 7 |
| -1 | 7 | 4 | 0 | 7 | -5 | 6 |
| 0 | -1 | 3 | 4 | 12 | 4 | 2 |
| | | | 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：

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

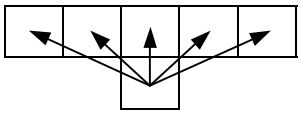


图 8

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：$F_{x,y}=\mathrm{Max}\{F_{x+2,y-1}, F_{x+1,y-1}, F_{x,y-1}, F_{x-1,y-1}, F_{x-2,y-1}\}+\mathrm{Num}_{x,y}$，其中 $\mathrm{Num}_{x,y}$ 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)。

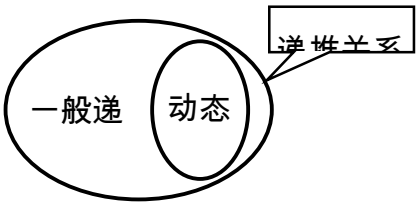



图 9

## 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 | $H_n=a_1\times H_{n-1}+a_2\times H_{n-2}+\cdots+a_r\times H_{n-r}$ | $H_n=\mathrm{Max}\{\cdots\}+a$ |

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》，很多人只是先利用公式 $C_n^r=\dfrac{n!}{r!(n-r)!}$ 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 $O(10^5\times10^5)$，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 |
|---|---|---|---|---|---|---|---|
| 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。