信息学奥林匹克竞赛指导丛书
吴文虎　主编

信息学奥林匹克竞赛指导
——1994—1995 竞赛试题解析

吴文虎　王建德　著

清华大学出版社

(京) 新登字 158 号

内 容 简 介

本书收集了 1994—1995 的国际国内有关信息学竞赛的试题及其解法分析，总数达 30 余道题。

这些试题有相当难度，是训练思维能力、提高解题技巧的很好的参考资料。本书内容的重点侧重于算法分析，力求取得最佳的算法。

本书是参赛选手的必读书，也可作为理工科大学生提高分析问题解决问题能力的参考书。

书　名：信息学奥林匹克竞赛指导
——1994—1995 竞赛试题解析
作　者：吴文虎　王建德
出版者：清华大学出版社(北京清华大学学研大厦，邮编 100084)
http://www.tup.tsinghua.edu.cn
印刷者：北京市人民文学印刷厂
发行者：新华书店总店北京科技发行所
开　本：787× 1092 1/16　印张：17　字数：402 千字
版　次：1997 年 4 月第 1 版　2002 年 8 月第 2 次印刷
书　号：ISBN 7-302-02238-0/TP・1089
印　数：5001～7000
定　价：18.00 元

前　言

国际信息学奥林匹克(International Olympiad in Informatics, IOI)从1989年到2001年,13年赛事的健康发展得益于联合国教科文组织(UNESCO)为这项赛事所做的准确定位:通过竞赛形式对有才华的青少年起到激励作用,促其能力得以发展;让青少年彼此建立联系,推动经验交流,给学校这一类课程增加活力;建立起教育工作者与专家档次上的国际联系,推进学术思想的交流。概括起来说,就是启迪思路,激励英才,发展学科,促进交流。

学科奥林匹克是智力与能力的竞赛,注重考查全面素质与创造能力。从这个意义上讲,信息学奥林匹克活动是素质教育的一个大课堂。在我国,每年国家集训队都要将“怎样做人,怎样做事,怎样求知和怎样健体”的指导思想纳入培训计划。这13年中国队共派出参赛选手51人次,累计获金牌26块、银牌14块、铜牌11块,届届名列前茅,正是因为坚持了全面素质教育的指导思想,把造就高素质有创造精神的人才作为活动的定位目标。

回顾13年的竞赛可以看出,参加高手云集的这种世界大赛是有相当难度的,第一,没有大纲,赛题范围没有界定,谁也无法去猜测每年的主办国会出什么类型的难题;第二,计算机科学与技术发展很快,层出不穷的新思路和新成果会反映到试题中来;第三,所要解决的试题往往涉及图论、组合数学、人工智能等大学开设的课程知识;第四,比较短的给定解题时间与刁难的测试数据让选手必须拿出高超和精巧的解法,无论在时间上还是空间上都是优化的解法才能取得高分。有许多赛题没有固定的现成的解法,选手要在比赛现场凭借实力,理出思路,构建数学模型,写出算法,编出程序,运行并验证整个构思是否正确,出解的时间是否能达到题目的要求,等等。可以看出,在这一过程中最重要的是要有创造能力。为激发创新精神,培养创造能力,就需要树立新的教育观念和教学方法,还要利用现代化的教学手段。引导学生学用电脑,在使用中帮助开发人脑,这可能是信息学奥林匹克活动的最重要的一个特点。我认为在这项活动中应该培养学生的四种能力:自学能力;实践动手能力;创新能力;上网获取信息,并能区分有用信息和无用信息的能力。这样做的结果使许多选手不但有能力在世界赛场上拿金牌,也有能力在学校的学习中名列前茅。

为了提高普及的层次,编写竞赛辅导教材是十分必要的。在本套丛书里我们将历年国际国内大赛的试题集中起来进行剖析,重点讲解题思路与方法。但是必须说明,书中的解法仅起抛砖引玉的作用。青少年是国家的希望,不断提高青少年的科学素养是中华民族永远昂首屹立在世界东方的根基所在。“精心育桃李,热望青胜蓝”是我和王建德老师的共同心愿。

国际信息学奥林匹克中国队总教练
清华大学计算机系教授博士生导师
吴文虎

2002年8月

重 印 说 明

科教兴国，提高我国青少年科学素养的重要性和迫切性已为人们所共识。相应的学科竞赛——信息学奥林匹克竞赛也为广大学生、家长和教师所关注。很多学生愿意参与这一类因材施教的课外活动，但又感到辅导教材奇缺。由清华大学出版社出版的这本教材，印数有限，很快就销售完了。我经常接到来信询问哪里能买到这类教材。为满足广大读者的需要，我们重新审读了这本书，认为其涉及的基本概念、基本原理、解题思路和解题技巧仍能对参加信息学奥林匹克竞赛活动的同学们起到积极的辅导作用。因此，我们选择了重印的方式，希望读者理解。

吴文虎

2002 年 8 月 9 日

喜　报

就在本书开机重印的前一天，我们惊喜地获悉：在 2002 年国际信息学奥林匹克竞赛(IOI 2002)中，中国队 4 名选手获得了三金一银的好成绩。同时 IOI 国际委员会授予吴文虎教授特别贡献奖，以表彰他为 IOI 所作出的突出贡献。

在此我们谨向 2002 年国际信息学奥林匹克竞赛中国队和吴文虎教授表示特别祝贺。

清华大学出版社

2002 年 8 月 29 日

目　　录

第一章　第六届国际信息学奥林匹克竞赛
中国组队赛试题分析

1.1　剔除多余括号

一、试题

键盘输入一个含有括号的四则运算表达式,可能含有多余的括号,编程整理该表达式,去掉所有多余的括号,原表达式中所有变量和运算符相对位置保持不变,并保持与原表达式等价。

例:输入表达式　　应输出表达式

a+(b+c)　　a+b+c

(a*b)+c/d　　a*b+c/d

a+b/(c-d)　　a+b/(c-d)

注意输入a+b时不能输出b+a。

表达式以字符串输入,长度不超过255。输入不要判错。

所有变量为单个小写字母。只是要求去掉所有多余括号,不要求对表达式化简。

二、算法分析

四则运算表达式含运算符+,-,*,/,(,),其优先顺序由低至高为

+,-→*,/→(,)

一个括号是否作为多余的括号剔除,关键是分析该括号内优先级最低的运算符与左邻括号或右邻括号的运算符之间的运算优先关系:

设待整理的表达式为(s_1 op s_2)

op—括号内优先级最低的运算符(+,-或*,/)

1.若左邻括号的运算符为/,则无论op为何运算符,括号必须保留,即…/(s_1 op s_2)…。因为原式(s_1 op s_2)作为除数,一旦去除括号后,仅s_1作为除数,不可能与原式等价。

2.若左邻括号的运算符为*或-,而op为+或-,则保留括号,即…*(-)(s_1 op s_2)。不能去除括号的原因是:

若左邻标符为*,优先级大于括号内运算符,因此必须用括号保证括号内的+或-优先进行;

若左邻算符为-,去除括号后,由于括号内的+、-要变号,结果不与原式等价。

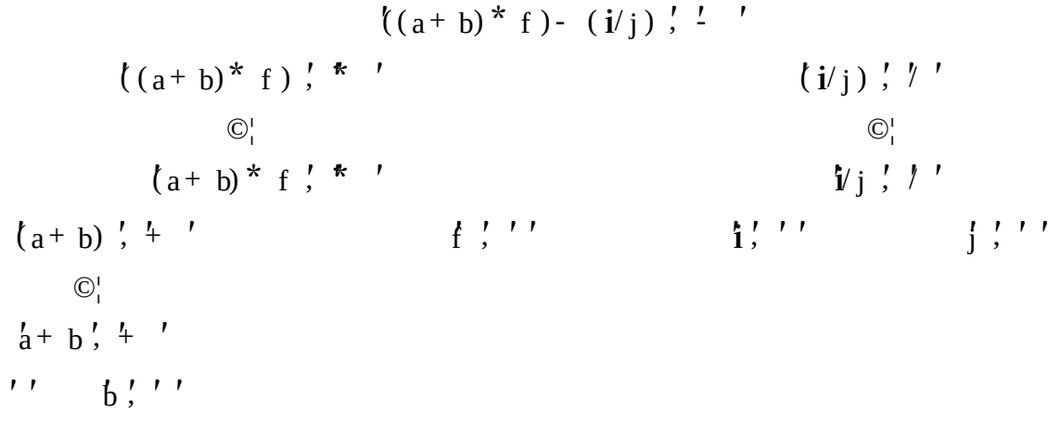
3.若右邻括号的运算符为*或/,而op为+或-,原式中的op运算必须优先进行,因此括号不能去除,即(s_1 op s_2)*/…。

4．除上述情况外，括号去除，结果与原式等价，即 …s_1 op s_2…。

我们从最里层嵌套的括号开始，依据上述规律逐步向外层进行括号整理，直至最外层的括号保留或去除为止。这个整理过程可以用下述的一个递归过程 reduce 描述：

```
function reduce(表达式串 s, var op) : string;
{ 整理表达式 s, 并返回 s 串中优先级最低的运算符 op 和整理结果 }
  begin
    if s 是变量 then
      begin
        op= ' '; reduce= 变量 s; exit
      end;
    if s 是(s')形式 then
      begin
        reduce= reduce(s', op); exit
      end;
  { s 是's1 op s2'形式 }
  a= reduce(s1, c1); b= reduce(s2, c2);
  if (op in ['*', '-']) and (c2 in ['+', '-']) then s2= '('s2')';
  if (op in ['*', '/']) and (c1 in ['+', '-']) then s1= '('s1')';
  if (op in ['/']) and (c2<>' ') then s2= '('s2')';
  reduce= 's1 op s2'
end;
```

例如，剔除表达式'((a+b)*f)-(i/j)'中多余的括号。依据上述算法进行整理的过程如下：



最后，自底向上得整理结果：(a+b)*f-i/j。

三、程序分析

```
program p11;

  var
    ins        :  string;        { 算术表达式 }
    p          :  char;          { 最低级运算符 }
```

```
function suitable(s:string;i:byte):byte;
  { s[i]为')',函数 suitable 输出与之相对应的'('在 s 中的位置 }
  var
    t:byte;
  begin
    t:=1;
    repeat
      dec(i);
      if s[i]=')'
        then inc(t)
        else if s[i]='('then dec(t)
    until t=0;
    suitable:=i
  end;   {suitable}

function find(s:string):byte;
  { 输出表达式 s 中最低级算符的位置 }
  var
    i,k:byte;
  begin
    i:=length(s);k:=0;
    while i>0 do
      begin
        if (s[i]='+') or (s[i]='-')
          then begin
                 find:=i;exit
               end; {then}
        if (k=0) and ((s[i]='*') or (s[i]='/')) then k:=i;
        if s[i]=')'then i:=suitable(s,i);
        dec(i)
      end; {while}
    find:=k
  end; {find}

function reduce(s:string;var p:char):string;
  { 整理表达式 s 并返回优先级最低的运算符 p }
  var
    a,b:string;    { 表达式 s 的左项和右项整理的结果 }
    c1,c2:char;    { 表达式 s 的左项和右项的最低级运算符 }
    i:byte;        { 表达式 s 的最低级运算符的位置 }
  begin
    if length(s)=1 { 若表达式是变量则返回变量 }
      then begin
```

```
            reduce:=s;p:='';exit
          end;{then}
      if (s[length(s)]=')') and (suitable(s,length(s))=1)
        {若表达式被一对括号包住,则返回括号内表达式的整理结果}
        then begin
          reduce:=reduce(copy(s,2,length(s)-2),p);
          exit
        end;{then}
      i:=find(s);p:=s[i];                    {找出最低级运算符及其位置}
      a:=reduce(copy(s,1,i-1),c1);           {分别整理左项和右项}
      b:=reduce(copy(s,i+1,length(s)-i),c2);
      {根据p和c1,c2,决定是否加括号}
      if (p in ['*','/']) and (c2 in ['+','-']) then b:='('+b+')';
      if (p in ['*','/']) and (c1 in ['+','-']) then a:='('+a+')';
      if (p='/') and (c2<>'') then b:='('+b+')';
      reduce:=a+p+b   {返回整理结果}
    end;{reduce}

  begin
    write('str=');
    readln(ins);              {输入算术表达式}
    writeln(reduce(ins,p))    {剔除多余括号并打印整理结果}
  end.{main}
```

1.2 瓷　　砖

一、试题

在 2N× 2N 的方格地板上铺放两种方形磁砖,表面图案分别为如图 1-1(a)和图 1-1(b)所示。一个方格中只可放一块磁砖,每种磁砖各有 $2N^2-1$ 块,加有两个方格内未铺任何磁砖。若相邻两块磁砖的弧线有共同点,则认为连起来的曲线是连续的。任何一种铺放方案其表面方案表现为若干曲线组成的图形。图 1-1(c)所示为 N=2 时的一种铺放方案,最长的连续曲线为由# 起始和终止的曲线。

对任何一块磁砖,若其上下左右 4 个方向某一相邻方格内没有磁砖,则可将该磁砖平移至该方格内。

图　1-1

编程要求:

对任何一种满足上述要求的铺放方案,寻找一种移动磁砖的方法,使得经过若干次移动后,所得其中的一条连续曲线在所有铺放方案中最长。

输入输出要求:

N 和初始铺放方案放在一个文本文件中。第一行是一个数字 N, 其下 2N 行为一个 2N× 2N 的矩阵, 若该矩阵第 I 行第 J 列为 1, 表示该方格处铺有形如图 1-1(a) 的磁砖, 为 2 表示铺有形如图 1-1(b) 的磁砖, 为 0 表示未铺任何磁砖。在上图这种铺放方案下的最长曲线的长度为 13, 但不一定是长度最长的方案。

N 为自然数, 1≤N≤30。

要求首先输出所求得的最长曲线的长度(一块磁砖上的一段曲线长度为 1), 然后将你的移动方案存入一名为 Solve. txt 的文件。该文件第一行为移动的总步数, 其下每一行表示一次移动。每行只含有 4 个数字“ x_1 y_1 x_2 y_2 ”, 表示当前棋盘上坐标(x_1, y_1) 处不是空格, 坐标(x_2, y_2) 处是空格, 且当前步是将在坐标(x_1, y_1) 的磁砖移至空格(x_2, y_2) 处, 显然(x_1, y_1) 与(x_2, y_2) 应当是相邻的。

备注:

(1) 不要求移动次数最少;

(2) 注意应能处理 N 值较大的情形;

(3) 最长曲线可以是封闭的, 也可以是不封闭的;

(4) 若有多条最长曲线只取一条;

(5) 输入文件不需判错。

二、算法分析

这个试题的算法即简单又复杂。所谓复杂, 即确定目标铺放方案非下一番功夫不可, 无捷径可走; 所谓简单, 即从任意铺放方案移动至目标方案, 除了有规律的机械移动。不需做任何算法上的“ 雕凿 ”。但程序要编写得正确、精练、易懂, 也不是一件简单的事。为了帮助读者理解, 我们不妨从下述 3 个方面展开讨论。

1. 空格移动

不言而喻, 空格有上、下、左、右 4 个移动方向, 如图 1-2 所示。

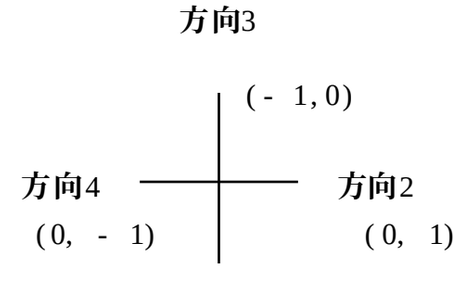


图 1-2

图中方向数为(垂直增量; 水平增量)

(x, y) 的空格沿 j(1≤j ≤4) 方向移动一格后的位置

x ← x+ 垂直增量

y ← y+ 水平增量

瓷砖平移至相邻的空格，相当于空格沿相反的方向移动一格，因此，我们可以将瓷砖平移看作是空格移动的过程。任一位置的瓷砖要移入某个指定的位置(i,k)，只有在两个空格有限次移动的配合下才能实现，其中空格 1 起着将该瓷砖引导至(i,k)的相邻格的“引路”作用，移动前位于(1,2n)；而空格 2 起“垫铺”作用，使得瓷砖在到达(i,k)的相邻格后可移入(i,k)。显然，移动前位于(i,k)。

为了表述空格的移动过程，我们定义了 3 个过程

(1) 空格(x,y)沿 j 方向移动 STEP 格

```
procedure move(x,y,j,step);
begin
  for i:= 1 to step do
    begin
      求(x,y)在 j 方向的相邻格(x',y')
      (x,y)←(x',y')的瓷砖
      x ← x'; y ← y'
    end;{for}
    (x,y) ← 空格;
end;{move}
```

(2) 空格(x_1,y_1)沿图 1-3 所示轨迹逆时针移动一周，使得轨迹上的瓷砖顺时针移动一格

```
Procedure left(x1,y1,x2,y2);
begin
  move(x1,y1,4,y1- y2);
  move(x1,y2,1,x2- x1);
  move(x2,y2,2,y1- y2);
  move(x2,y1,3,x2- x1);
end;{left}
```

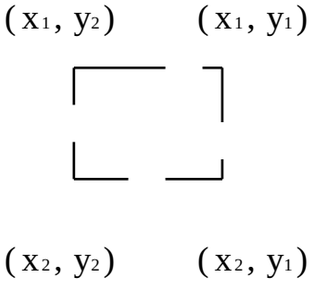


图 1-3

(3) 空格(x_1,y_1)沿如图 1-4 所示轨迹顺时针移动一周，使得轨迹上瓷砖逆时针移动一格

```
procedure right(x1,y1,x2,y2);
begin
  move(x1,y1,1,x2- x1);
  move(x2,y1,4,y1- y2);
  move(x2,y2,3,x2- x1);
  move(x1,y2,2,y1- y2);
end;{right}
```

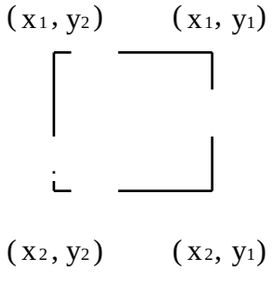


图 1-4

2. 目标状态的确定

由于有两个空格,所以任何两个瓷砖铺放方案都能通过有限次的移动相互转换,因而最好预先确定目标铺放方案,这样可以使得搜索沿即定的方向进行,避免盲目性。

我们定义最终目标铺放方案如图 1-5 所示。

图 1-5

图 1-6

当 n= 1 时

显然曲线长度为 2;

当 n= 2 时的目标铺放方案如图 1-6 所示。

曲线长度为 19,

除 n= 1, 2 外的目标铺放方案为

两个空格分别位于(1, 2n) (2n, 2n);

含图 1-1 (a) 图案瓷砖的方格包括:

第 1 行的(1, 5) (1, 7) … (1, 2n- 1) 以及(1, 2) (1, 4);

第 2 行的(2, 6) (2, 7) … (2, 2n) 以及(2, 1) (2, 2) (2, 4);

第 3 行的(3, 3) (3, 4);

第 4, 6, 8, …, 2(n- 1) 行所有方格;

第 2n 行的(2n, 1) (2n, 3) … (2n, 2n- 3)。

除上述方格外的所有方格铺图 1-1(b) 图案瓷砖。

这样定义的铺放方案从第(1, 2(n- 1)) 出发,从上到下,往返迂回至(2n, 2(n- 1)),曲线走向的规律如下:

(1) 除周边外,中间的$(2(n-1))^2$块瓷砖每块通过两段曲线$(8n^2-16n+8)$

(2) 位于周边的起始端和结束端的两块瓷砖各通过二段曲线(2× 2= 4)

(3) 其余瓷砖各通过一段曲线(8n- 8)

这样的曲线总长度合计为$8n^2-8n+4$。

例如 n= 3 时的目标铺放方案如图 1-7 所示。

显然曲线长度为 52。

图 1-7

由于试题不要求移动次数最少,因此只要给出一种机械移动的过程即可。为了便于移动,将最终目标铺放方案的瓷砖顺次往下移动一格,使得两个空格分别位于(1, 2n) 和(2, 2n),形成中间目标铺放方案。然后着手将初始铺放方案移动至中间目标方案。

3. 搜索过程

搜索前先将初始铺放方案中的第 1 个空格移至右上角(1, 2n)、第 2 个空格移至第 1 个搜索位置(1, 1), 然后按先列后行、从上而下、从左至右的顺序, 逐格地将初始铺放方案移动成中间目标铺放方案, 搜索顺序如图 1-8 所示。

搜索的轨迹为 2(n- 1)条, 每条轨迹 k(1≤k≤2(n- 1)) 如图 1-9 所示的搜索线, 即先由上而下搜索 k 列上的 1 至 2n- k 行位置, 后由左至右搜索 2n- k+ 1 行的 k 至 2n 列位置。

图 1-8

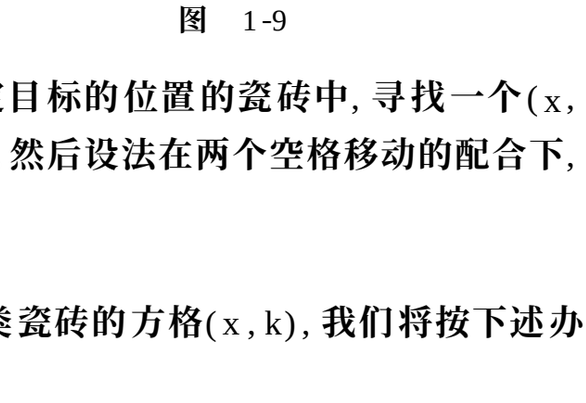


图 1-9

若沿轨迹搜索至(i, k)位置时, 则在当前未确定目标的位置的瓷砖中, 寻找一个(x, y), 该瓷砖与中间目标铺放方案中的(i, k)瓷砖同类。然后设法在两个空格移动的配合下, 将(x, y)的瓷砖移至(i, k)。具体过程如下:

(1) 由上而下搜索 k 列时

若当前 k 列有一个与中间目标方案中(i, k)同类瓷砖的方格(x, k), 我们将按下述办法将该瓷砖垂直移到(i, k)

```
while x> i+ 1 do{循环, 直至(x, k)的瓷砖上移至(i+ 1, k)}
begin
  move (1, 2n, 1, x- 2);      {将空格 1 下移至(x- 1, 2n)}
  left (x- 1, 2n, x, k);       {瓷砖上移一格}
  move (x- 1, 2n, 3, x- 2);    {空格 1 返回(1, 2n)}
  x ← x- 1;                    {行指针上移}
end;                           {While}
move (i, k, 1, 1);             {通过空格 2 下移, 将瓷砖移至(i, k)}
```

此时, 空格 2 正好落在下一个搜索位置(i+ 1, k)上。

若垂直搜索过程中, 同类瓷砖(x, y)不在 k 列(x≠k), 我们先借助于空格 1, 将(x, y)瓷砖垂直移至(i, y)

```
if y= 2n then
begin
  left( 1, 2n, x, 2n- 1);
  y←y- 1;
```

```
end   {将(x,2n)的瓷砖左移至(x,2n-1)}
While x<i do      {循环,将位于i行上方的该瓷砖下移至(i,y)}
   begin
     right(1,2n,x+1,y);
     x←x+1;
   end;   {while}
While x>i do      {循环,将位于i行下方的该瓷砖上移至(i,y)}
   begin
     left(1,2n,x,y);
     x←x-1;
   end;   {while}
```

接着，再在空格1的配合下将(i,y)的瓷砖左移至(i,k+1)

```
if x>1 then while y>k+1 do
  begin left(1,2n,x,y-1);
        y←y-1;end {while}
  else while y>k+1 do
    begin right(1,2n,2,y-1);
          y←y-1;end {while}
```

然后通过 move(i,k,2,1)，将位于(i,k) 的空格2左移1格，使得同类瓷砖最终移入(i,k)

最后分析(i,k)位置，若非垂直搜索位置(i≤2n-k)，则通过

move(i,k+1,1,1)；move(i+1,k+1,4,1)

将空格2由(i,k+1)移至下一搜索位置(i+1,k)，继续沿轨迹垂直搜索。

(2) 由左至右搜索2n-k+1行

若当前行有一个瓷砖(x,y)与目标铺放方案中(2n-k+1,i)的瓷砖同行(x=2n-k+1)，我们则按下述方法将之水平移到(2n-k+1,i)

```
while y>i+1 do {循环,直至(2n-k+1)的瓷砖左移至(2n-k+1,i+1)}
   begin
     left(1,2n,x,y-1);{瓷砖左移一格}
     y ← y-1;         {列指针左移}
   end;               {while}
move(2n-k+1,i,2,1);{通过空格2右移,将瓷砖移入(2n-k+1,i)}
```

若水平搜索过程中，同类瓷砖(x,y)不在当前行(x≠2n-k+1)，我们先借助于空格1将(x,y)的瓷砖下移至(2n-k,y)

```
if y=2n the begin left(1,2n,x,y-1);
y ← y-1;end; {将(x,2n)的瓷砖左移至(x,2n-1)}
while x<(2n-k) do {循环,直至(x,y)的瓷砖下移至(2n-k,y)}
   begin
```

```
        right (1, 2n, x+ 1, y); x←x+ 1; {瓷砖下移一格, 行指针下移}
    end; {while}
```

接着再借助于空格 1 将(2n- k, y)的瓷砖水平移至(2n- k, i)

```
while y< i do begin right (1, 2n, x, y); y←y+ 1; end; {若在 i 列左端, 则右移}
while y> i do begin left(1, 2n, x, y- 1); y←y- 1; end; {若在 i 列右端, 则左移}
```

然后通过 move (x + 1, y, 3, 1), 将位于(2n- k+ 1, i)的空格 2 上移 1 格, 使得同类瓷砖最终移入(2n- k+ 1, i)。

最后将位于(2n- k, i)的空格 2 移至下一搜索位置

```
if i= 2n then begin {若当前行搜索完, 则空格 2 移至(1, k+ 1)开始新一轮
                     垂直搜索}
                  move (x, y, 4, y- k- 1); move (x, k+ 1, 3, x- 1);
                end {then}
           else begin {否则, 将空格 2 移至(2n- k+ 1, i+ 1); 继续当前的水平
                     搜索}
                  move (x, y, 2, 1); move (x, y+ 1, 1, 1);
                end; {else}
```

继续沿轨迹搜索。

(3) 将中间目标铺放方案转换成最终目标方案

经上述步骤后, 空格 1 位于(1, 2n), 空格 2 位于(1, 2n- 1)。要确定位于(2, 2n)最后一块瓷砖的移动方向:

如果当前(2, 2n)的瓷砖类同于中间目标铺放方案中(2, 2n- 1)的瓷砖, 我们则通过

```
move (1, 2n- 1, 1, 1); move (2, 2n- 1, 2, 1)
```

在空格 2 的配合下, 将(2, 2n)的瓷砖左移至(2, 2n- 1);

否则当前(2, 2n)的瓷砖类同于中间目标铺放方案中(1, 2n- 1)的瓷砖。我们通过

```
move (1, 2n, 1, 1); move (1, 2n- 1, 2, 1)
```

在两个空格的配合下, 将(2, 2n)的瓷砖移入(1, 2n- 1)。

这样一来, 便完成了向中间目标铺放方案移动的过程。最后, 由

```
move (2, 2n, 1, 2n- 2)
```

将(2, 2n)的空格移至右下方(2n, 2n), 将之转换为最终目标铺放方案。

三、程序分析

下面, 我们采用逐步求精的办法给出程序题解

1. 初始化

第一层

```
program longest_pattern_among_bricks;
  1.1 定义和说明;
    begin
  1.2 init;   {输入初始铺放方案}
```

2. 计算和输出移动方案

```
  1.3 make_obj; {确定目标铺放方案}
      assign(f,'brick.sol'); rewrite(f);
  1.4 init_blank; {将初始铺放方案中的两个空格分别移至左上角和右上角}
  1.5 main;   {计算和输出从初始铺放方案至目标铺放方案的移动过程}
      close(f);   {关闭文件}
    end. {main}
```

第二层

1．求精 1.1——定义和说明

```
const
  way                 : array[1..4,1..2] of integer
                      = ((1,0),(0,1),(-1,0),(0,-1));
        {way[i,1]——i 方向的垂直增量;way[i,2]——i 方向的水平增量; 1≤i≤4}
var
  sub,obj             : array[1..60,1..60] of integer;
        {sub——初始铺放方案;obj——目标铺放方案;}
  n                   : integer;   {地板规模为 2n*2n}
  f                   : text;   {文件变量}
```

2．求精 1.2——init 的过程说明

```
procedure init;
  var filename : string; {文件名串}
      i,j      : integer;
  begin
    write('input file name : ');readln(filename); {读入文件名串}
    assign(f,filename);reset(f);   {文件名与文件变量连接,读准备}
    readln(f,n);              {读入地板规模}
    for i:=1 to 2*n do  {顺序读入 2N*2N 的矩阵}
      for j:=1 to 2*n do
        read(f,sub[i,j]);
    close(f);   {关闭文件}
end;   {init}
```