奥林匹克计算机竞赛试题剖析

吴文虎　刘福生　王建德 编著

跳棋
多项式与列表转换
求最长公共子串
合并表格
八进制数除法
求最长路径
最佳旅行路线

AOLINPIKEJISUANJI
JINGSAISHITI
POUXI

上海科技教育出版社

奥林匹克计算机竞赛试题剖析

吴文虎　刘福生　王建德　编著

上海科技教育出版社

内 容 提 要

本书针对1993年全国第十届青少年信息学竞赛、中国组建第五届国际信息学参赛队选拔赛及第五届国际奥林匹克信息学竞赛等三大赛题进行剖析，以每次赛题为一单元，逐题解释题意、详细阐述解题的思路和方法，并对程序求精作详细分析。第四章给出三大赛题的全部源程序，读者可对其进行编译运算。

奥林匹克计算机竞赛试题剖析

吴文虎　刘福生　王建德　编著

上海科技教育出版社出版发行

（上海冠生园路393号　邮政编码 200233）

各地新华书店经销　江苏省丹阳市教育印刷厂印刷

开本 850×1168　1/32　印张 7.375　字数 185000

1995年12月第1版　1995年12月第1次印刷

印数 1－3200

ISBN7-5428-1126-6/G·906

定价：9.70元

培奧賽強手

育世紀新才

張敦祥

一九九七年四月

前　言

第五届国际奥林匹克信息学竞赛(IOI'93)于 1993 年 10 月 24 日在南美阿根廷的门多萨拉开了战幕，参加这次国际逐鹿的中国代表队经选拔确定由郭远山(福建师大附中)、黄天明(广西南宁二中)、张辰(北京师大附属实验中学)、柴晓路(上海控江中学)等四名同学组成，他们远征西半球，奋力拼搏，以取得一金(郭远山)、一银(黄天明)、两铜(张辰、柴晓路)的优异成绩为祖国赢得了荣誉。

国际奥林匹克信息学竞赛是联合国科教文组织发起的，目前已有 46 个国家参加这一世界性的青少年计算机知识大赛。自 1989 年至今国际奥林匹克信息学竞赛已历经五届，中国队在吴文虎领队统率下，五次组队参赛，前后选派 18 名队员披挂出征，共获奖牌 18 枚，其中金牌 7 枚，银牌 5 枚、铜牌 6 枚，并创下全体队员都获奖的奥赛纪录。我国广大青少年深受历次胜利的鼓舞，人人摩拳擦掌，大有跃跃欲试之势，参赛人员有的来自沿海大城市，也有的来自老、少、边、穷地区，这说明，只要我们努力和重视，任何地方的青少年都可以脱颖而出，为国扬鞭海外，策马驰骋在计算机知识的领地里。

为了满足我国青少年了解有关国际奥林匹克信息学竞赛选拔及国际大赛的情况，也为了向全国各地在选拔和培训选手过程中提供最新的信息和资料，我们特地编写了这本书，从中国队组队选拔直到 1993 年的国际大赛，本书列选了全部赛题，并作了详细的剖析和介绍。同时，我们也希望能为一切关心国际奥林匹克信息学竞赛、关心青少年计算机教育及所有喜爱计算机程序设计的人们送上一册有益的参考书籍。

我们衷心希望我国的青少年一代能在各项事业中胜过他们的父辈，抢占现代科学的前沿，用他们的双手和脑子把我国建设成为四个现代化的强国。

本书的主要读者应是广大爱好计算机的青少年，鉴于我国中学计算机课程已开始(或准备)讲授结构程序设计语言 PASCAL，所以我们在书中亦采用 PASCAL 语言来描述过程。在剖析每道试题时，都力求使读者明确试题的要求，并着重说清思维过程，以期读者能举一反三。我们在交代清楚赛题的含义以后，便作算法分析，详细讲述解决问题的思路，紧接着是程序的求精分析，三段叙述一气呵成。此书既可用作各地各学校国际奥林匹克信息学竞赛(计算机)培训和集训教材，又可作为学校 PASCAL 语言教学的补充读物。如读者需要对以往各届国际奥林匹克信息学竞赛赛题的解题技巧作进一步了解时，请参阅著者 1992 年编著的《青少年国际信息学(计算机)奥林匹克竞赛指导》一书。

本书承蒙上海市计算机技术研究所副总工程师王景寅研究员指导审定，我们在此深表谢意。

作　者

1993 年 11 月

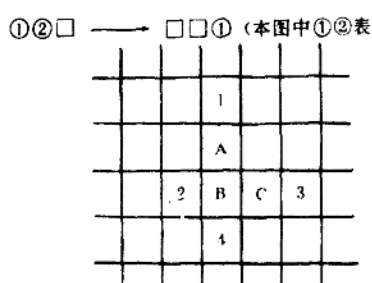
目　录

第一章　第五届国际奥林匹克信息学竞赛中国组队选拔赛试题解析

第一节　跳　　棋

一、试题

在一个矩阵中定义一种跳棋的规则，它的基本类型见图 1.1-1：



（图 1.1-1）

但棋 1 只能向上方、左方、右方跳动吃掉棋子。

如图 1：棋 B 可跳至 3 位吃掉棋 C；棋 C 可跳至 2 位吃掉棋 B；棋 B 可跳至 1 位吃掉棋 A；但棋 A 不可跳至 4 位吃掉棋 B。

现要求实现以下两个任务：

任务 1：

寻求一个初始状态，使之经过一系列走步后，使只有第一行有一枚棋子。

规定一个棋盘，宽度不大于 8，长度不知有多少。定义一个 n（$0 \leqslant n \leqslant 4$），使初始状态中的前 n+1 行无棋子。

要求最佳（棋子数最少）。

任务 2：

寻求一个初始状态，使之经过一系列走步后，使只有第一行的两枚中有 m 个空格的棋子（$0 \leqslant m \leqslant 6$）。

规定一个棋盘，宽度不大于 8，长度不知有多少。定义一个 n（$0 \leqslant n \leqslant 4$），使初始状态中前 n+1 行无棋子。

要求最佳（棋子数最少）。

二、算法分析

1. 设定目标状态

设有棋行数为 length，即前 length 行有棋；右端棋子的列位置为 side；

对于任务 1 的目标状态来说，我们指定[1,1]有一枚棋子，其他棋格为空。

即 length=1，side=1；

对于任务 2 的目标状态来说，我们分别指定[1,1]，[1,m+2]有两枚棋子，其他棋格为空。

即 length=1，side=m+2；

我们从目标状态出发，采用倒推手法，寻找最佳的初始状态。

2. 求沿某方向倒推一步后的状态

用 x，y 分别记棋盘的行坐标和列坐标。

设[x,y]为倒推出发位置，即棋子是沿某方向跳至[x,y]的，

而我们则从[x,y]出发,沿该方向的逆方向倒推跳前状态。根据跳棋规则,显然在跳前状态中,[x,y]的棋子数减1,沿途经过的其他两位置的棋子数加1。但问题是:

① 若沿→方向走至[x,y]且y<3,则倒推前先将棋盘中所有棋子右移3−y列,即设定棋子沿→方向跳至[x,3]。从[x,3]出发,沿→的逆方向倒推跳前状态。

② 若沿↑方向走至[x,y]且x+2>length,则倒推后length变为x+2;

③ 若沿←方向走至[x,y]且y+2>side,则倒推后side变为y+2。

3. 求倒推一步的最佳方向

设倒推位置为[x,y]。若[x,y]=0(即无棋),按分析2显然是不可能倒推跳前状态的,退出(递归边界)。否则从[x,y]倒推。要倒推一个最佳的跳前状态之关键是要弄清:从未来求出的棋子数最少的初始状态出发,跳若干步后,棋子沿哪个方向跳至[x,y]。

首先在x+2≤纵深极限的前提下假设↑方向,并计算出[x+1,y]、[x+2,y]两位置的棋子和作为当前最佳代价。若x+2>纵深极限,程序失败退出。

但是否选择↑方向为最佳方向,还要根据←、→方向倒推的情况确定:

① 若允许选择←方向,且经过位置的棋子和少于当前最佳代价,则分别递归求出沿其他方向跳至两个经过位置的最少棋子数C1、C2。

若C1+C2亦少于当前最佳代价,则设定←方向为最佳方向,C1+C2为当前最佳代价;

② 若允许选择→方向且经过位置的棋子和少于当前最佳代价,则分别求出沿其他方向跳至两个经过位置的最佳代价C1、C2。若C1+C2亦少于当前最佳代价,则设定→方向,C1+C2为当前

最佳代价。但是要注意两个特例：

i. 若 y=1，由→方向未经棋格，显然 C1=0、C2=0，因此设定→方向为最佳方向和当前最佳代价为 0 是勿庸质疑的了；

ii. 若 y=2，由→方向仅经过[x,1]，只要[x,2]的棋子数少于当前最佳代价且递归求出的跳至[x,1]的最佳代价 C1(C2=0)亦少于当前最佳代价，则设定→方向为最佳方向，当前最佳代价为 C1。

4. 从目标状态出发倒推初始状态

目标状态中仅第一行有棋，其他行无棋。我们从第 1 行出发，按下列算法逐步处理前 n+1 行：

```
for  times:=1  to  n+1  do
  begin
    求沿↑方向跳至 times 行的所有棋子的跳前状态，使得
  前 times 行无棋；
    repeat
    搜索 times+1..length 行的所有棋格，选择一个在棋子
  数大于 1 的有棋格中棋子数最少的位置[x,y]，根据算法分析
  3 求倒推一步的最佳方向 best-way。(若具有最少棋子数的有
  棋格有多个，则分别运用算法分析 3 求倒推一步的最佳代价，
  从中确定倒推代价最少的位置[x,y]和由它倒推一步的最佳
  方向 best-way)；
    根据分析 2 求从[x,y]出发、沿 best-way 方向倒推一步
  后的跳前状态；
    until times+1...length 行的所有有棋格的棋子数为 1；
  end；{for}
```

注意：为了防止无穷搜索下去，我们设立一个檻值——若在倒推过程中发现棋子总数大于 20，则认定不可能有解，无解退出。

三、程序的求精分析

第一层-

1. 定义和说明

```
uses
  crt;
const
  maxlength = 1000;{ 纵深极限 }
  maxwidth = 8;    { 宽度极限 }
  maxn = 8;        { n 的极限  }
type
  boardtype = array [ 1.. maxlength, 1..
              maxwidth] of integer;
  { 棋盘矩阵 }

var
  board: boardtype;
  {board[i,j]——[i,j]位置的棋子个数 }
  n,m,width: integer;
  length,          { 前 length 行有棋子 }
  side: integer;   { 右端棋子所在的列 }
  tot : integer;   { 棋子总数 }
```

2. 求解问题 1

```
begin
  writeln('***problem 1***');
  2.1 init(1);  { 问题 1 的初始化 }
  2.2 move;     { 求解 }
```

3. 求解问题 2

```
    writeln('***problem 2***');
    2.1 init(2);   { 问题 2 的初始化 }
    2.2 move;      { 求解 }
  end.
```

第二层

1. 求精 2.1 —— init(1)的过程说明。

```
procedure  init(code:integer);
{ 输入及初始量的赋值。参数 code ——问题代码 }
var  i,j:integer;
begin
  repeat
    write('width=');{ 输入宽度 }
    readln(width);
  until  (width>0)  and  (width<=maxwidth);
  repeat{ 定义一个 n,要求初始状态中前 n+1 行无棋子 }
    write('n=');
    readln(n);
  until  (n>=0)  and  (n<=maxn);
  for  i:=1  to  maxlength  do{ 棋盘初始化 }
    for  j:=1  to  width  do  board[i,j]:=0;
  board[1,1]:=1;   {[1,1]有一枚棋子,即目标状态 }
  length:=1;       { 目标状态仅第一行有棋子 }
  case  code  of
  1: begin
    tot:=1;        { 目标状态棋子总数为 1 }
      side:=1;     { 目标状态右端棋子的列为 1 }
    end;
```

```
    2: begin
      repeat
        write('m=');
        { 输入目标状态中两枚棋子间的空格数 }
        readln(m);
      until (m>=0) and (m<=width-2);
      board[1,m+2]:= 1 ; { 第二枚棋子位于[1,m+2] }
      tot:= 2 ;
      { 目标状态有两枚棋子 }
      side:=m+ 2 ;
      { 目标状态右端棋子的列为 m+2 }
      end;
  end;
end;
```

2. 求精 2.2 —— move 的过程说明。

```
procedure  move;    { 求解 }
var
  times,
  small,
  {当前所有位置中棋子数最少的值,简称最少棋子数}
  {从小于或等于最小棋子数的位置倒推,求最佳跳前状态}
  i,j,
  best-cost,best-way,
  { 最佳代价——在跳动过程中经过的最少棋子数, 最佳方
    向 }
  x,y,cost,way : integer;
  { 倒推跳前状态的最佳位置,代价和方向 }
begin
```

```
for  times:=1  to  n+1  do
{ 逐行处理前 n+1 行,使得前 n+1 行无棋子 }
begin
2.2.1 clear-first-line(times);
{ 求沿↑方向跳至 times 行的所有棋子的跳前状态 }
repeat
  small:=maxlength;      { 棋子最少数初始化 }
  { 下面搜索棋子数最少的最佳位置[x,y]以及倒推跳前
    状态的最佳方向 best-way }
  for  i:=times+1  to  length  do
    for  j:=1  to  side  do
    {从 times+1 行开始搜索所有含棋子的位置}
      if  (board[i,j]>1)  and  (board[i,j]<=small)
       then
       {可以从[i,j]倒推,且该位置的棋子数最少}
        if  board[i,j]<small  then
        { 若[i,j]的棋子数小于当前最少棋子数,则该位
          置棋子数为最少棋子数,并求[i,j]倒推的最佳
          方向 best-way,确定[i,j]为最佳倒推位置 }
        begin
          small:=board [i,j];
    2.2.2   find-way-to-shoot(i,j.side,true,true,best-
cost,best-way);
          x:=i;   y:=j;
        end
  else
  { 否则[i,j]的棋子数等于当前的最少棋子数,则求出[i,j]
    倒推的最佳代价 cost 和最佳方向 way,若 cost 小于当前
```

```
        最佳代价 best-cost,则确定 cost 为当前最佳代价,way
        为当前最佳方向,[i,j]为最佳倒推位置 }
    begin
      find-way-to-shoot(i,j,side,true,true,cost,way);
      if  cost<best-cost  then
      begin
        best-cost:=cost;
        best-way:=way;
        x:=i;   y:=j;
      end;
    end;
    if  small<maxlength
    { 若存在最佳倒推位置[x,y],则从[x,y]开始,沿 best-way
      方向倒推跳前状态 }
    2.2.3 then  shoot(x,y,best-way);
        if  tot>20  then  begin
        { 若棋子数大于 20,输出无解,否则输出棋盘状态 }
          clrscr;
          writeln('no  answer!');
          exit;
        end
    2.2.4 else  show;
        until  small=maxlength;
        { 直至无最佳倒推位置[x,y]为止,即 times 行后每个有
          棋格仅有一枚棋子 }
      end;
    end;
```

第三层

1. 求精 2.2.1 —— clear-first-line(times)的过程说明。

```
procedure   clear-first-line(line:integer);
{ 求沿↑方向走至 line 行的所有棋子的走前状态 }
var
  i:integer;
begin
  for  i:=1   to   width   do  { 逐列分析 }
    if   board[line,i]=1
{ 若[line, i]仅含一枚棋子,则沿↑的逆方向倒推跳前状态 }
       then   shoot(line,i,0);
end;
```

2. 求精 2.2.2 —— find-way-to-shoot(i,j,side,true,true,best-cost,best-way)的过程说明。

```
procedure   find-way-to-shoot(
  x,y,e:integer;
  { x,y ——倒推位置; e ——目前右端棋子的位置 }
  right,left:boolean;
  { 向左、右跳动标志,若值为 false,则不允许沿该方向跳动}
var   best-cost,
  { 跳动过程中经过的最少棋子数,即最佳代价,过程调用
    后,结果需返回主程序 }
  best-way:integer);
  { 最佳方向数(-1 ——→; 0 ——↑; 1 ——←)。从[x,
    y]开始,沿该方向的逆方向倒推,经过的棋子数最少。过
    程调用后,结果需返回主程序 }
var
```