

# 基于栈结构的孔明棋算法研究

张桂芬,葛丽娜,黄银娟

(广西民族大学 数学与计算机科学学院,广西 南宁 530006)

**摘 要:**孔明棋是一种玩法简单,但其中变化无数的益智游戏。对孔明棋求解问题进行分析,提出了基于回溯思想的递归和非递归算法,运行结果表明了算法的有效性。文章还围绕栈在存储数据、消解递归等方面的应用对两个算法的优缺点进行了比较分析,递归算法结构清晰,但递归调用次数多;而非递归算法借助程序栈,将程序向循环转化,降低了时间复杂度,但算法难以分析和理解。因此在求解实际问题时可以采用递归思想来分析,然后借助栈用非递归来实现算法。

**关键词:**孔明棋;栈结构;递归;非递归;回溯

**中图分类号:**TP301.6

**文献标识码:**A

**文章编号:**1673-629X(2009)12-0051-04

## Research of Kongming Chess Algorithm Based on Stack - Structure

ZHANG Gui-fen, GE Li-na, HUANG Yin-juan

(College of Mathematics & Computer Science, Guangxi University for Nationalities, Nanning 530006, China)

**Abstract:** Kongming chess is an intellectual game with simple rules but changeable playing measures. Backtracking is an important and efficient solution for many issues. On the analysis of Kongming chess, proposes recursion and non-recursion algorithms based on backtrack for the issue, which works effectively. At the same time, advantages and disadvantages of the two algorithms are compared in the stack-based applications for data storage, backtracking problem, and recursion. Recursion has a vivid algorithm structure, but it has to make use of recurring for many times; While non-recursion circulates the program with the help of a program stack. Thus it lowers the complexity of time but it has a complicated algorithm which is hard to analyse and understand. It comes to a conclusion that recursion is suitable for analysis, while non-recursion is for algorithm in solving practical problems.

**Key words:** Kongming chess; stack-structure; recursion; non-recursion; backtracking

## 0 引言

栈是限定仅在表尾进行插入或删除操作的线性表,具有“后进先出”的特性,广泛应用在各种软件系统中,是算法设计重要的一种数据结构<sup>[1]</sup>。文中对孔明棋求解问题提出了基于栈结构的递归和非递归算法,给出了算法的具体实现过程,分析栈的应用,实验结果验证了方法的有效性。

## 1 提出问题

孔明棋,也叫单身贵族、独立钻石棋,传说是三国时代孔明发明的益智棋。它与华容道、魔术方块同被称为智力游戏界的三大不可思议。孔明棋是单人就可以玩的游戏,由33颗棋子排成井字型盘面,取去中央的那个棋子,开始展开游戏,如图1所示。游戏玩法似

中国跳棋游戏,将棋子跳过邻近的棋子,到达一个旁边空着的位置,被跳过的棋子则从棋盘上取去,跳的路径可以是上、下、左、右,但不可斜跳,直到剩下最后一颗棋子在棋盘中央,游戏便胜利结束,否则游戏失败(剩余多颗棋子或剩余一颗棋子但不在棋盘中央)。该游戏玩法非常简单,但其中变化无数,成为风靡世界的智力游戏。求解一次取胜的移棋步骤成为人们期待解决的问题。

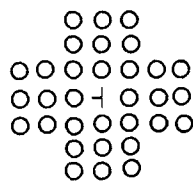


图1 孔明棋的初始棋局

## 2 分析问题

### 2.1 数据结构的选择

利用计算机求解孔明棋一次取胜的移棋步骤,可以采用试探和回溯的搜索方法,即从初始棋局开始,对

收稿日期:2009-03-11;修回日期:2009-07-02

基金项目:广西自然科学基金资助项目(0832084)

作者简介:张桂芬(1974-),女,广西凌云人,讲师,硕士,研究方向为算法设计。

所有棋子的所有方向依次进行移棋探索,若能移,则移动棋子并继续下一新棋局的探索;否则撤销移棋,回溯到上一棋局,继续依次进行移棋探索,直到所有棋子的所有方向都探索到为止<sup>[2]</sup>。为了保存从初始棋局到当前棋局的所有移棋步骤,可以

使用栈结构、数组、文件等作为存储结构,本算法移棋和撤棋是以后进先出方式进行的,所以选择了栈。数据结构如下:

棋局: typedef char kongmiqitype [7]

[7]; /\* \* '表示棋子; ' + '表示空

位; - '表示非法位置

当前棋子位置: typedef struct {

int i; //行坐标

int j; //列坐标

} postype; [3]

移棋步骤: typedef struct {

int ord; //移棋步骤的序号

postype seat; //棋子的坐标位置

int di; //移棋方向, 0、1、2、3 分别表示棋子的右、下、左、上方向

向

} selemtype; //栈的元素类型

栈结构: typedef struct {

selemtype \* base; //栈底指针

selemtype \* top; //栈顶指针

int stacksize; //栈的分配存储空间

} sqstack;

## 2.2 算法分析

从初始棋局开始,每走一步会少一颗棋子,得到一个新棋局,最后剩一颗棋子在棋盘中央便可取胜<sup>[4]</sup>。用栈存储移棋步骤,假设当前棋局指“在搜索过程中某一时刻的棋局”,则求解孔明棋的算法思想是:若当前棋局有棋子可移动,则移棋且“移棋步骤”进栈,并继续下一个棋局的探索,即切换它成为当前棋局,如此反复直到求得解为止;若当前棋局所有棋子都不能移棋,则撤棋回到上一棋局并将“移棋步骤”出栈,朝后续的棋子方向探索。对每个棋局的探索范围都是从第一颗棋子(0 行 0 列)的第一个方向(0 方向)到最后一颗棋子(6 行 6 列)的第四个方向(3 方向)。当探索成功结束,从栈底到栈顶依次存储着孔明棋一次取胜的移棋步骤<sup>[5]</sup>。根据分析,图 2 展示了孔明棋求解过程中棋局状态的变化情况。

状态树中每个结点表示一个棋局,“树根”是初始棋局,而所有的“叶子”就是可能出现的结局,它可能剩余一颗或多颗棋子,而只有“剩下最后一颗棋子在棋盘中央”的叶子才是求解的结局。据此,问题求解的本质即为在约束条件下先根遍历状态树的过程,对每一个

结点,判断是否求解的结局,如是则输出并返回,如不是,则依次先根遍历满足约束条件的各棵子树,即判定该子树根是否能移棋,若能,则先根遍历该子树,否则剪去该子树分支<sup>[2]</sup>。

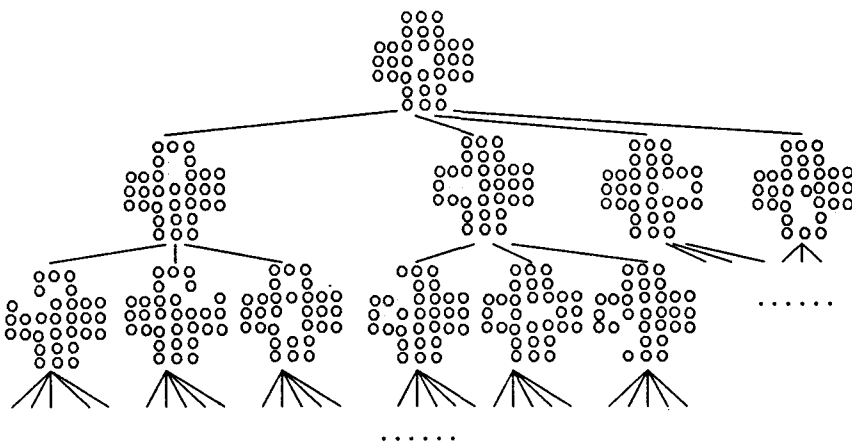


图 2 孔明棋求解过程的棋局状态树

## 3 栈与递归算法

### 3.1 算法实现

递归是实现回溯的一种重要方式。递归算法的设计是对问题的抽象过程,如果抽象到每个小问题都有相同特征时,那就形成了递归<sup>[6]</sup>。

据前分析,孔明棋的回溯求解状态树具有递归特性,算法实现如下:

```
status kongminqpath(int curstep){
    if (victory(kmq)) {printstack(s); flag = true; return(true);} //
    取胜,并设置退出所有递归标志位
    else{
        for(i=0;i<=6; ++i)
            for(j=0;j<=6; ++j)
                for (di=0;di<=3;di++)
                    if (pass(kmq,i,j,di)) {/(i,j)位置的棋子能向 di 方
                    向移动
                        move(kmq,i,j,di); //移棋
                        e.ord = curstep; e.seat.i = i; e.seat.j = j; e.di = di;
                        push(s,e); //移棋步骤进栈
                        kongminqpath(curstep+1); //探索下一棋局
                        if (flag == true) return(ok); //退出所有递归
                        pop(s,e); //移棋步骤出栈
                        back(kmq,i,j,di); //撤棋
                    }; //endif
                }; //endif
        return(false); //问题无解
    }; //kongminqpath
```

在主函数中设置 flag 初值为 false,调用 kongminqpath(1),可求解出一个取胜的移棋步骤序列:

1:(1,3),1→2:(2,1),0→3:(0,2),1→  
 4:(0,4),2→5:(2,3),2→6:(2,0),0→  
 7:(2,4),3→8:(2,6),2→9:(3,2),3→  
 10:(0,2),1→11:(3,0),0→12:(3,2),3→  
 13:(3,4),2→14:(3,6),2→15:(3,4),3→  
 16:(0,4),1→17:(4,2),3→18:(1,2),1→  
 19:(4,0),0→20:(4,3),2→21:(5,4),3→  
 22:(4,6),2→23:(6,2),3→24:(3,2),1→  
 25:(6,4),2→26:(6,2),3→27:(4,1),0→  
 28:(4,3),0→29:(2,4),1→30:(4,5),2→  
 31:(5,3),3

### 3.2 栈的应用

在递归算法中是由系统来设置和管理“递归工作栈”的,在每次执行递归调用语句之前,自动把子程序中所使用的值参 curstep 和局部变量 i,j,di 的当前值以及调用后的返回地址进栈,在每次递归调用结束后,又自动把栈顶元素的值分别赋给相应的值参和局部变量,以便使它们恢复到调用前的值,接着无条件转向由返回地址所指定的位置继续执行子程序,使 back() 函数正确撤棋,回溯到上一棋局。递归算法的“保存现场”和“恢复现场”是完全利用栈的“后进先出”特性来实现的。本算法中除了系统工作栈外,还设置了程序管理的栈 s,它只起到存储数据(移棋步骤序列)的作用。

栈的代码:move(kmq,i,j,di); //移棋

```
e.ord = curstep; e.seat.i = i; e.seat.j = j; e.di = di;
push(s,e);
.....
pop(s,e);
back(kmq,i,j,di); //撤棋
```

### 3.3 效率分析

不难看出,本递归算法结构清晰、层次感强,给程序的阅读和调试带来很大的方便。但此程序递归和重复调用次数多,问题求解共需递归调用 20277 次,调用开销大。所需辅助空间:一为探索过程中系统工作栈的最大容量,为 32 层(状态树深度)“现场数据”的存储空间;二为程序栈 s 存储的 31 个移棋步骤的存储空间。

## 4 用栈改递归为非递归算法

### 4.1 算法实现

递归过程转换成非递归过程的实质是将原来由系统管理的“递归工作栈”改为由程序来管理,一般会借助栈和循环结构来实现转换。转换算法为:

(1)设置栈 s,用于存储移棋步骤,同时也记录了

求解过程中的回溯点。

(2)处理当前的元素,若能移棋(pass(kmq,i,j,di)=true)则设置回溯点,将该回溯点进栈 s 保存(push(s,e))。

(3)如果求解已达到目标(victory(kmq)=true),则算法以成功告终;如果当前求解动作无法继续进行(flag=false),而此时栈 s 又已为空,则算法以失败告终;否则取出栈顶元素(pop(s,e)),进行回溯,并转(2)执行<sup>[7]</sup>。

具体算法为:

```
status kongminqpath(kongmiqitype &kmq){
    curi=0; curj=0; curdi=0; curstep=1; //设置((0,0),0)为第
    1 个棋局探索的初始位置与方向
    do {flag=false; //退出多重循环的标志位
        for(i=curi; ((i<=6)&&(!flag)); ++i){
            for(j=curj; ((j<=6)&&(!flag)); ++j){
                for (di=curdi; ((di<=3)&&(!flag)); di++){
                    if (pass(kmq,i,j,di)){
                        move(kmq,i,j,di); //移棋
                        e.ord=curstep; e.seat.i=i; e.seat.j=j; e.di=
di;
                        push(s,e); //移棋步骤进栈
                        if (victory(kmq)) {printstack(s); return
(true);} //取胜
                        curstep++;
                        flag=true; //结束三重循环,探索下一棋局
                    } //endif
                    curdi=0; //重置方向初始值
                } //endj
                curj=0; //重置列初始值
            } //endi
            curi=0; //重置行初始值
            if (flag=false) //当前棋局所有棋子都不能移棋
                if (!stackempty(s)){
                    pop(s,e); //移棋步骤出栈
                    back(kmq,e.seat.i,e.seat.j,e.di); //撤棋
                    curstep--;
                    curi=e.seat.i; curj=e.seat.j; curdi=e.di+1; //换后续的
                    棋子方向探索
                } //endif
            } while (!stackempty(s));
            return(false); //问题无解
        } // kongminqpath
```

主函数初始化 kongmiqitype 类型变量 kmq 为初始棋局,然后调用函数 kongminqpath(kmq),运行结果与递归算法的相同。

### 4.2 栈的应用与效率分析

栈的代码:move(kmq,i,j,di); //移棋

```
e.ord=curstep; e.seat.i=i; e.seat.j=j; e.di=di;
push(s,e);
.....
pop(s,e);
back(kmq,e.seat.i,e.seat.j,e.di); //撤棋
curi=e.seat.i; curj=e.seat.j; curdi=e.di+1;
```

在两个算法中,进栈部分的代码是相同的,栈  $s$  以清晰的  $e(\text{ord}, \text{seat}, \text{id})$  格式存储着移棋步骤。而出栈的代码则不同,由于栈  $s$  在递归算法中存储的移棋步骤信息正好也是回溯点所需保存的信息,所以本算法在非递归转化过程中,并不需要再增加新的栈,但此时栈  $s$  已经具有两个作用:一是提供撤棋函数  $\text{back}()$  的参数以及恢复现场变量  $\text{curi}, \text{curj}, \text{curdi}$  的值,实现棋局回溯;二是保存移棋步骤序列。通过这一比较,就很好理解“递归算法非递归化一般都是借助栈来实现”这一设计方法了。

非递归算法不使用系统栈,利用递归与循环算法的内在规律,借助程序栈,将程序向循环转化,算法执行过程不依赖于函数或过程的重复调用,大大降低了时间复杂度。本算法所需辅助空间只是程序栈  $s$  的存储空间,空间复杂度降低了。

非递归算法的缺点是程序复杂且难以分析和理解。因此在求解实际问题时可以采用递归思想来分析,然后用非递归来实现算法<sup>[8]</sup>。

## 5 结束语

文中针对孔明棋求解方式的特点,以栈为数据结构,利用回溯方法实现了递归和非递归算法,实验结果验证了方法的有效性,对同类问题的算法设计具有一定指导和参考意义。本算法还可在多方面进行研究和探讨,如求解孔明棋的所有解、探索路径的优化等。

### 参考文献:

- [1] Ford W, Topp W. Data Structures with C++ [M]. [s.l.]: Prentice Hall, 1996.
- [2] 严蔚敏, 吴伟民. 数据结构(C语言版)[M]. 北京: 清华大学出版社, 2006.
- [3] 辛玲, 王相海. 国际象棋中马的周游路线问题的递归算法[J]. 计算机工程与设计, 2006, 27(1): 47-48.
- [4] 李立中, 林顺喜. 孔明棋的电脑解法研究[EB/OL]. 2002-08-23. <http://alg.ice.ntnu.edu.tw/icewise/docs/Kon-Ming.doc>.
- [5] Kruse R, Tondo C L, Leung B. Data Structures & Program Design in C[M]. [s.l.]: Prentice-Hall, 1997.
- [6] Sahni S. Data Structures Algorithms and Applications in C++ [M]. Columbus: Mc Graw-Hill, 1999.
- [7] 杨庆红, 罗坚. 递归问题的非递归实现方法研究与应用[J]. 计算机时代, 2005(8): 44-45.
- [8] 马军红. 递归与非递归算法比较及效率分析[J]. 科技信息, 2008(31): 554-556.

(上接第 50 页)

测试表明, PNS-Grid 算法在系统运行初期, 节点每次转发都将触发对路由表的优化操作, 针对性强, 且每次都测量较多节点, 优化幅度较大, 而后节点路由表能以较快的速度被调整到最优值, 并且达到优化开销较小的状态。

### 参考文献:

- [1] 杨文俊. P2P 网络系统中节点自组织管理机制[J]. 计算机技术与发展, 2006, 16(7): 57-59.
- [2] Stoica I, Morris R, Karger D, et al. Chord: A scalable peer-to-peer lookup service for Internet applications[R]. [s.l.]: MIT, 2001.
- [3] Bolosky W, Douceur J, Ely D, et al. Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs [C] // Proceedings of SIGMETRICS 2000. Santa Clara, CA: [s.n.], 2000.
- [4] Waldman M, Rubin A D, Cranor L F. Publius: A robust, tamper-evident, censorship-resistant, web publishing sys-

tem[C] // Proceedings of the 9th USENIX Security Symposium (August 2000). Berkely: IEEE Press, 2000: 59-72.

- [5] Aberer K. P-Grid: A self-organizing access structure for P2P information systems[C] // Proc of IFLP International Conference on Network and Parallel Computing Workshop. Los Alamitos: IEEE Computer Society Press, 2007: 437-441.
- [6] Sandberg R. The Sun Network Filesystem: Design, Implementation and Experience[C] // Proceedings of the 1987 Summer Usenix Conference. New York: ACM Press, 1987: 300-314.
- [7] Oceanstore project home page[EB/OL]. 2006. <http://oceanstore.cs.berkeley.edu>. BLACK M.
- [8] Druschel P, Rowstron A. PAST: A large-scale, persistent peer-to-peer storage utility[M]. HotOS VIII, Schloss Elmau, Germany: [s.n.], 2001: 75-80.
- [9] 张庆丰, 李东琦, 唐慧佳. 基于 P2P 分布式文件传输系统的研究[J]. 微计算机信息, 2007, 23(3): 92-94.
- [10] 赵慧娟, 王汝传. 基于遗传算法的 P=2P 资源发现算法[J]. 南京邮电大学学报: 自然科学版, 2007, 27(4): 85-89.