**姓名: 周丙瑞**

**学号: E11514058**

# 【人工智能实验报告】

# 树式搜索算法求解八皇后问题

【实验目的】

1. 学会用状态图描述抽象问题

2. 运用树式状态图搜索算法求解实际问题, 加深对树式搜索算法的理解

【实验原理】

1. 搜索

搜索（Search），设法在庞大状态空间图中找到目标。

主要分为两类性质的搜索：

基本搜索，是一种没有任何经验和知识起作用的、由某种规则所确定的非智能性的搜索；

启发式搜索（Heuristic Search）：其特点在于是一种有准备的、追求效率而有的放矢的智能搜索，它要求依据某种知识及信息的指导，通过逐一状态比较而找到符合规定条件的目标状态解。

2. 问题的状态空间图搜索

问题的状态空间可用有向图来表达，又常称其为状态树(State Tree)。图中，节点Sk表示状态，状 态之间的连接采用有向弧(Arc)，弧上标以操作数OK来表示状态之间的转换关系。

用状态空间法搜索求解问题:

首先要把待求解的问题表示为状态空间图;

把问题的解表示为目标节点Sg。

求解就是要找到从根节点S0到达目标节点Sg的搜索路径。

3. 状态空间表示法

状态空间

状态，描述某一类事物在不同时刻所处于的信息状况

操作，描述状态之间的关系

问题的状态空间可用一个三元序组来表示：

：问题的全部初始状态的集合

：操作的集合

：目标状态的集合

4.用树式广度优先算法求解八皇后问题

状态表示

数据结构 ：

一个一维数组存已经放置皇后的行皇后所在列数;

三个bool数组分别存当前状态的列冲突、两方向斜线的冲突;

一个整数标记当前装态深度(已放置皇后行数)

状态转换规则

当前状态扩展子节点时, 根据深度值寻找下一行任意一个满足条件(不被冲突)的位置放置皇后

算法描述

　　　１) 初始节点放入OPEN表

２) 若OPEN为空, 退出, 输出八皇后方案数

３) 移出OPEN表中第一个节点放入CLOSED表

　　　４) 若当前节点深度为８(目标节点), 输出当前节点状态(八皇后位置)，转(2)

　　　５）根据当前节点的深度和冲突, 扩展新的节点加入CLOSED表, 转(2).

实验输入输出

输入　:　 无

　　输出　: 每一个八皇后的状态(用01矩阵表示, 皇后用１表示)、总的方案数

实验细节

编码过程中, 关于状态的冲突表示中

一条斜线冲突的斜率为１, 用横纵坐标差值表示该冲突，要注意防止负值出现

【实验代码】

#include <cstdio>

#include <cstring>

using namespace std;

struct Node

{

int a[10];

bool col[10], line[18], tline[18];

//a存放当前行放置的皇后位置

//col数组标记父节点列冲突, line和tline标记两条斜线的冲突

Node()

{

memset(a, 0, sizeof a);

memset(col, 0, sizeof col);

memset(line, 0, sizeof line);

memset(tline, 0,sizeof tline);

}

};

struct OpenNode

{

Node node;

int fa; //父节点编号

int dep; //当前节点已放置的皇后深度(初识为0)

OpenNode \* next;

OpenNode(){}

OpenNode(int f, int d, Node newNode)

{

node = newNode;

fa = f;

dep = d;

next = NULL;

}

};

struct ClosedNode

{

OpenNode node;

int id;

ClosedNode \* next;

ClosedNode(OpenNode \* v, int idnum)

{

id = idnum;

node = \*v;

next = NULL;

}

};

namespace EightQueens

{

ClosedNode \* closedList = NULL; //Closed表

OpenNode \* openList = NULL; //Open表

int cnt = 0; //用于生成编号

void addClosed(ClosedNode \* node) //添加到Closed表中

{

if(closedList == NULL)

{

closedList = node;

return;

}

ClosedNode \* tmp = closedList;

while(tmp->next) tmp = tmp->next;

tmp->next = node;

}

void addOpen(OpenNode \* node) //添加到Open表中

{

if(openList == NULL)

{

openList = node;

return;

}

OpenNode \* tmp = openList;

while(tmp->next) tmp = tmp->next;

tmp->next = node;

}

void bfs() //广度优先搜索

{

OpenNode \* init = new OpenNode(0, 0, Node()); //初识节点

openList = init;

int ans = 0; //方案数

while(openList)

{

OpenNode \* now = openList;

openList = openList->next; //移动Open表

ClosedNode \* tmp = new ClosedNode(now, ++cnt); //加入Closed表并编号

addClosed(tmp);

if(tmp->node.dep == 8) //放置完成,输出皇后

{

++ans;

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

for(int i = 1; i <= 8; ++i)

{

for(int j = 1; j <= 8; ++j)

printf("%d ", tmp->node.node.a[i] == j ? 1 : 0);

printf("\n\n");

}

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");

}

else

{

int currentRow = tmp->node.dep + 1; //扩展节点

for(int i = 1; i <= 8; ++i)

if(!now->node.col[i] && !now->node.line[i+currentRow] && !now->node.tline[7+i-currentRow]) //列与斜线

{

OpenNode \* temp = new OpenNode(tmp->id, now->dep + 1, now->node);

temp->node.a[currentRow] = i;

temp->node.col[i] = 1;

temp->node.line[i + currentRow] = 1;

temp->node.tline[7 + i-currentRow] = 1;

addOpen(temp);

}

}

}

printf("%d\n", ans);

}

}

int main()

{

EightQueens::bfs();

return 0;

}

【实验结果】

