姓名: 周丙瑞

学号: E11514058

【人工智能实验报告】

　　A\*算法求解八数码问题

【实验目的】

1. 加深对A\*算法的理解

2. 运用A\*算法求解实际问题

【实验原理】

１.算法：Ａ算法广度优先搜索

２.用Ａ算法求解八数码问题

八数码状态表示:

空格用数字９表示, 以便于用一个整数唯一标识一个状态

eg:

2 1 3

4 5 9

7 6 8

表示当前空格(9)在第二行第三列, 且可用213459768唯一标识该状态

用一个map<int, pair<int, int> > 记录每一个数字的位置

eg:

5位于第二列第二行

　 3.h(x)和g(x)

g(x):初始节点到当前节点的转移的步数

　　　 h(x):当前节点到目标节点的曼哈顿路径之和

4.状态转换规则

当前状态扩展子节点:

1) 先依据当前状态的map找到空格(数字９)的位置

2) 根据四个移动方向, 判断边界合法后转移, 同时修改转移后的数字位置和map

3) 根据八数码标识的整数判断新的状态是否已被访问, 若未被访问, 记录该状态的父 节点, 计算f(x)值并加入OPEN表(OPEN表按f值升序列排序)

５.算法描述

　　　１) 初始节点放入OPEN表

２) 若OPEN为空, 退出

３) 移出OPEN表中第一个节点放入CLOSED表

　　　４) 若当前节点＝目标节点, 则搜索成功, 输出路径, 结束

　　　５）根据当前节点的状态, 扩展新的节点加入CLOSED表, 转(2).

６.实验输入输出

　 输入:八数码初始状态(9表示空格)、目标状态

　　 输出:从初始状态到目标状态的路径

７．实验细节

1)标识父节点用状态唯一标识

2)输出路径采用回溯的实现方式

【实验代码】

#include <cstdio>

#include <cstring>

#include <utility>

#include <map>

using namespace std;

typedef pair<int, int> PII;

typedef map<int, int> MII;

int dx[] = {-1, 0, 1, 0}; //初始化方向数组

int dy[] = {0, -1, 0, 1};

inline int Abs(int x) { return x >= 0 ? x : -x; }

struct Node

{

int num[3][3]; //八数码各位上数字

PII mp[10]; //各个数字对应的位置

int getV() const //用一个整数标记状态

{

int ans = 0;

for(int i = 0; i < 3; ++i)

for(int j = 0; j < 3; ++j)

ans = 10 \* ans + num[i][j];

return ans;

}

int getDis(const Node &rhs) const //h函数(两状态各数字曼哈顿路径之和)

{

int res = 0;

for(int i = 0; i < 3; ++i)

for(int j = 0; j < 3; ++j)

res += Abs(mp[num[i][j]].first - rhs.mp[num[i][j]].first) + Abs(mp[num[i][j]].second - rhs.mp[num[i][j]].second);

return res;

}

bool operator == (const Node &rhs) const

{

return getV() == rhs.getV();

}

void in()

{

for(int i = 0; i < 3; ++i)

for(int j = 0; j < 3; ++j)

scanf("%d", &num[i][j]), mp[num[i][j]] = make\_pair(i, j); //初始化

}

void Print() //输出节点

{

printf("\n");

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

for(int i = 0; i < 3; ++i)

{

for(int j = 0; j < 3; ++j)

printf("%d ", num[i][j]);

puts("");

}

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

}

};

struct OpenNode

{

Node node;

OpenNode \* next;

int fa; //父节点(以数值为ID)

int g, f; //代价级估价函数值

OpenNode(){}

OpenNode(Node \_node, int \_fa, int \_g)

{

node = \_node;

fa = \_fa;

g = \_g;

next = NULL;

}

};

struct ClosedNode

{

OpenNode node;

ClosedNode \* next;

ClosedNode(OpenNode \* \_node)

{

node = \* \_node;

next = NULL;

}

};

namespace EightFigure

{

OpenNode \* openHead = NULL;

ClosedNode \* closedHead = NULL;

Node s, e; //起始节点和目标节点

MII vis; //vis标记(因为数值较大用map)

map<int, ClosedNode \* > father; //用整数标记的父节点ID映射一个父节点

void Input()

{

printf("输入初识状态: \n");

s.in();

printf("输入目标状态: \n");

e.in();

}

void addClosed(ClosedNode \* p)

{

if(closedHead == NULL)

{

closedHead = p;

return;

}

ClosedNode \* tmp = closedHead;

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

tmp->next = p;

}

void addOpen(OpenNode \* p)

{

if(openHead == NULL || openHead->f > p->f)

{

p->next = openHead;

openHead = p;

return;

}

OpenNode \* tmp = openHead;

while(tmp->next)

{

if(tmp->f <= p->f && tmp->next->f > p->f) break;

tmp = tmp->next;

}

p->next = tmp->next;

tmp->next = p;

}

bool judge(int x, int y) //判边界

{

if(x < 0 || x >= 3 || y < 0 || y >= 3) return false;

return true;

}

void PrintPath(ClosedNode \* p) //回溯

{

int f = p->node.fa;

if(f != -1) PrintPath(father[f]);

p->node.node.Print();

}

void Work()

{

openHead = new OpenNode(s, -1, 0);

vis[s.getV()] = 1;

while(openHead != NULL)

{

ClosedNode \* tmp = new ClosedNode(openHead);

addClosed(tmp);

openHead = openHead->next;

if(tmp->node.node.getV() == e.getV()) //找到目标节点

{

printf("\n\n路线: \n");

PrintPath(tmp);

return;

}

PII ninePos = tmp->node.node.mp[9];

for(int i = 0; i < 4; ++i) //扩展

{

int x = ninePos.first, y = ninePos.second;

int fx = x + dx[i];

int fy = y + dy[i];

if(judge(fx, fy)) //先判边界

{

Node newNode = tmp->node.node;

swap(newNode.num[x][y], newNode.num[fx][fy]);

swap(newNode.mp[newNode.num[x][y]], newNode.mp[newNode.num[fx][fy]]);

if(!vis[newNode.getV()]) //判重

{

OpenNode \* newOpenNode = new OpenNode(newNode, newNode.getV(), tmp->node.g + 1);

father[newNode.getV()] = tmp;//映射父节点

newOpenNode->f = newOpenNode->g + newNode.getDis(e);

addOpen(newOpenNode);

vis[newNode.getV()] = 1; //标记

}

}

}

}

}

void Solve()

{

Input();

Work();

}

}

int main()

{

EightFigure::Solve();

return 0;

}

【实验结果】

