实验五 A\*算法解决八数码问题

171491328 罗晓璐

问题描述：通过单步移动把下面的矩阵移动成1-8环绕一周的矩阵（即0在中间，1-8顺序排成一圈，1在哪无所谓）

|  |  |  |
| --- | --- | --- |
| 2 | 8 | 3 |
| 1 | 6 | 4 |
| 7 | 0 | 5 |

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 8 | 0 | 4 |
| 7 | 6 | 5 |

初始状态 目标状态

假设启发式的方程为f(n)=d(n)+h(n)，其中d(n)为层次或深度，h(n)为错误的个数，使用启发式算法解决。

启发式搜索就是在状态空间中对每一个状态进行评估，找到最好的状态，再从这个状态出发直至到达目标状态。每次寻找最佳的状态可以省略大量不需要的搜索，提高了效率。

启发式搜索的算法如下：

a) 把初始节点放入Open表中，计算其f值；

b) 如果Open表为空，则问题无解，失败退出；

c) 把Open表的第一个节点取出放入Close表，并标记该节点为n；

d) 考察n节点是否为目标节点。如果是，则得到问题的解，成功退出；

e) 如果节点n不可扩展，则转第b)步；

f) 扩展节点n，计算每一个子节点的f值，并为每个子节点设置指向节点n的指针，将这些子节点放入Open表中；

g) 根据各节点的f值，对Open表中给的全部节点按照从小到大的顺序排序；

h) 转第b)步。

代码：

#include <iostream>

#include <queue>

#include <stack>

#include <vector>

#include <algorithm>

#include <memory.h>

using namespace std;

// 八数码状态

typedef struct \_Status{

int status[3][3];

\_Status \*parent;

\_Status \*next;

}Status;

// AStar排序依据

bool decComparator(const Status &s1, const Status &s2){

int gn1 = 0, gn2 = 0;

int dn1 = 0, dn2 = 0;

const Status \*ptr1 = &s1;

const Status \*ptr2 = &s2;

int status[3][3] = {1,2,3,8,0,4,7,6,5};

while(ptr1 != NULL){

gn1 += 1;

ptr1 = ptr1->parent;

}

while(ptr2 != NULL){

gn2 += 1;

ptr2 = ptr2->parent;

}

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

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

if(s1.status[i][j] != status[i][j]){

dn1 += 1;

}

if(s2.status[i][j] != status[i][j]){

dn2 += 1;

}

}

}

return (gn1+dn1) > (gn2+dn2);

}

// 八数码搜索

class EightPuzzle{

private:

unsigned char allHash[362880];

Status root;

Status goal;

private:

int nextNumber;

Status next[4];

public:

EightPuzzle(Status \*root, Status \*goal){

memcpy(&this->root.status, &root->status, sizeof(int)\*9);

this->root.parent = NULL;

this->root.next = NULL;

memcpy(&this->goal.status, &goal->status, sizeof(int)\*9);

this->goal.parent = NULL;

this->goal.next = NULL;

}

private:

// 判断是否是目标状态

inline int IsGoal(Status \*tmp){

return memcmp(&tmp->status, &goal.status, sizeof(int)\*9);

}

// 下一个可行的状态

int NextStatus(Status \*tmp){

nextNumber = 0;

int posi, posj;

for(int i = 0; i < 9; i++){

posi = i/3, posj = i - i/3\*3;

if(tmp->status[posi][posj] == 0){

break;

}

}

if(posi-1 >= 0){

Status left = \*tmp;

left.status[posi][posj] = left.status[posi-1][posj];

left.status[posi-1][posj] = 0;

if(allHash[Cantor(left.status)] == 0){

next[nextNumber] = left;

next[nextNumber].parent = tmp;

nextNumber++;

}

}

if(posi+1 <= 2){

Status right = \*tmp;

right.status[posi][posj] = right.status[posi+1][posj];

right.status[posi+1][posj] = 0;

if(allHash[Cantor(right.status)] == 0){

next[nextNumber] = right;

next[nextNumber].parent = tmp;

nextNumber++;

}

}

if(posj-1 >= 0){

Status up = \*tmp;

up.status[posi][posj] = up.status[posi][posj-1];

up.status[posi][posj-1] = 0;

if(allHash[Cantor(up.status)] == 0){

next[nextNumber] = up;

next[nextNumber].parent = tmp;

nextNumber++;

}

}

if(posj+1 <= 2){

Status down = \*tmp;

down.status[posi][posj] = down.status[posi][posj+1];

down.status[posi][posj+1] = 0;

if(allHash[Cantor(down.status)] == 0){

next[nextNumber] = down;

next[nextNumber].parent = tmp;

nextNumber++;

}

}

return nextNumber;

}

// 康托展开

int Cantor(int arr[][3]){

int fac[10] = {1,1,2,6,24,120,720,5040,40320,362880};

int index = 0;

for(int i = 7; i >= 0; i--){

int irow = i/3, icol = i - i/3\*3;

int count = 0;

for(int j = 8; j > i; j--){

int jrow = j/3, jcol = j - j/3\*3;

if(arr[jrow][jcol] < arr[irow][icol]){

count++;

}

}

index += (count\*fac[8-i]);

}

return index;

}

public:

int AStar(){

int step = 0;

memset(allHash, 0, 362880);

vector<Status> openTable;

Status \*closeTable = new Status;;

Status \*current = closeTable;

Status \*tmp;

openTable.push\_back(root);

allHash[Cantor(root.status)] ;

while(!openTable.empty()){

tmp = new Status;

\*tmp = openTable[openTable.size()-1];

openTable.pop\_back();

step++;

current->next = tmp;

current = current->next;

if(IsGoal(tmp) == 0){

PrintPath(tmp);

freeCloseTable(closeTable);

return step;

}

int nextNumber = NextStatus(tmp);

if(nextNumber == 0){

continue;

}

for(int i = 0; i < nextNumber; i++){

openTable.push\_back(next[i]);

allHash[Cantor(next[i].status)] ;

}

sort(openTable.begin(), openTable.end(), decComparator);

}

cout << "AStar failed." << endl;

freeCloseTable(closeTable);

return -1;

}

private:

// 打印路径

void PrintPath(Status \*head){

if(head == NULL){

return;

}

else{

PrintPath(head->parent);

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

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

cout << head->status[i][j]<<" ";

}

cout << endl;

}

cout <<endl;

}

}

// 释放close表

void freeCloseTable(Status \*closeTable){

Status \*current;

while(closeTable != NULL){

current = closeTable->next;

free(closeTable);

closeTable = current;

}

}

};

int main()

{

Status init = {2,8,3,1,6,4,7,0,5,0,NULL};

Status goal = {1,2,3,8,0,4,7,6,5,0,NULL};

EightPuzzle ep = EightPuzzle(&init, &goal);

cout << "AStar\*\*\*\*\*\*\n" << endl;

cout << "step: " << ep.AStar() << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\n" << endl;

return 0;

}

程序结果：

