****

**中国地质大学（武汉）**

**人工智能导论实验报告**

**2021 年12月18日**

# 八数码问题的A\*搜索算法实现

#### 一 题目要求：

要求：设计估价函数，并采用c++编程实现，以八数码为例演示A\*算法的搜索过程，争取做到直观、清晰地演示算法，代码要适当加注释。

八数码问题：在3×3方格棋盘上，分别放置了标有数字1,2,3,4,5,6,7,8的八张牌，初始状态S0根据题目要求设定，使用的操作有:空格上移，空格左移，空格右移，空格下移。试采用A\*算法写程序实现这一搜索过程。

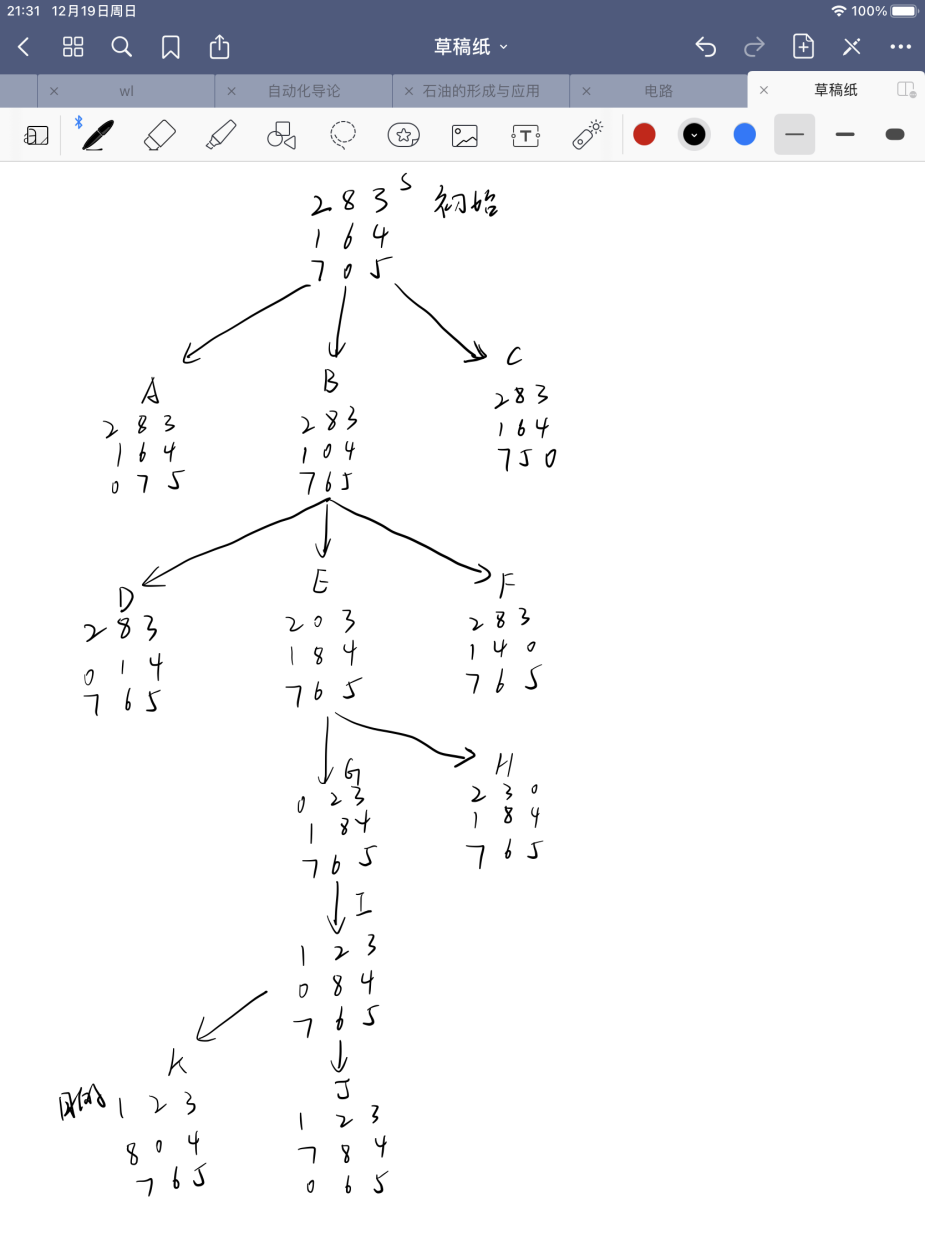
#### 二 实验要求：

1. 设置相同的初始状态和目标状态,针对不同的估价函数,求得问题的解,比较它们对搜索算法性能的影响,包括扩展节点数、生成节点数等,填入表1。
2. 设置与上述1相同的初始状态和目标状态,用宽度优先搜索算法(即令估计代价h(n)=0的A\*算法)求得问题的解,以及搜索过程中的扩展节点数、生成节点数,填入表1。

表1不同启发函数h(n)求解8数码问题的结果比较

|  |  |  |  |
| --- | --- | --- | --- |
|  | 启发函数*h***(***n***)** | | |
| 不在位数 | 将牌“不在位”的距离和 | 0 |
| 初始状态 | 283164705 | | |
| 目标状态 | 123804765 | 123804765 | 123804765 |
| 最优解 | 2 8 3 1 6 4 7 0 5  ↓  2 8 3 1 0 4 7 6 5  ↓  2 0 3 1 8 4 7 6 5  ↓  0 2 3 1 8 4 7 6 5  ↓  1 2 3 0 8 4 7 6 5  ↓  1 2 3 8 0 4 7 6 5 | 2 8 3 1 6 4 7 0 5  ↓  2 8 3 1 0 4 7 6 5  ↓  2 0 3 1 8 4 7 6 5  ↓  0 2 3 1 8 4 7 6 5  ↓  1 2 3 0 8 4 7 6 5  ↓  1 2 3 8 0 4 7 6 5 | 2 8 3 1 6 4 7 0 5  ↓  2 8 3 1 0 4 7 6 5  ↓  2 0 3 1 8 4 7 6 5  ↓  0 2 3 1 8 4 7 6 5  ↓  1 2 3 0 8 4 7 6 5  ↓  1 2 3 8 0 4 7 6 5 |
| 扩展节点数  （不包括叶子节点） | 5 | 5 | 37 |
| 生成节点数  （包含叶子节点） | 11 | 11 | 60 |
| 运行时间  （迭代次数） | 6 | 6 | 38 |

#### 三 画出[2, 8, 3], [1, 6, 4], [7, 0, 5]推导至[1, 2, 3], [8, 0, 4], [7, 6, 5]的图解。



#### 四 分析不同的估价函数对A\*算法性能的影响

①不在位数结果

请输入初始状态...

2 8 3 1 6 4 7 0 5

请输入目标状态...

1 2 3 8 0 4 7 6 5

--------------------

起始状态逆序数为 11 为奇

目标状态逆序数为 7 为奇

至少要移动5步!

2 8 3

1 6 4

7 0 5

g: 当前深度：0

h: 启发函数：4

f=g+h=4

2 8 3

1 0 4

7 6 5

g: 当前深度：1

h: 启发函数：3

f=g+h=4

2 0 3

1 8 4

7 6 5

g: 当前深度：2

h: 启发函数：3

f=g+h=5

0 2 3

1 8 4

7 6 5

g: 当前深度：3

h: 启发函数：2

f=g+h=5

1 2 3

0 8 4

7 6 5

g: 当前深度：4

h: 启发函数：1

f=g+h=5

1 2 3

8 0 4

7 6 5

g: 当前深度：5

h: 启发函数：0

f=g+h=5

②哈密顿距离结果

请输入初始状态...

2 8 3 1 6 4 7 0 5

请输入目标状态...

1 2 3 8 0 4 7 6 5

--------------------

起始状态逆序数为 11 为奇

目标状态逆序数为 7 为奇

至少要移动5步!

2 8 3

1 6 4

7 0 5

g: 当前深度：0

h: 启发函数：5

f=g+h= 5

open表长6

2 8 3

1 0 4

7 6 5

g: 当前深度：1

h: 启发函数：4

f=g+h= 5

open表长6

2 0 3

1 8 4

7 6 5

g: 当前深度：2

h: 启发函数：3

f=g+h= 5

open表长6

0 2 3

1 8 4

7 6 5

g: 当前深度：3

h: 启发函数：2

f=g+h= 5

open表长6

1 2 3

0 8 4

7 6 5

g: 当前深度：4

h: 启发函数：1

f=g+h= 5

open表长6

1 2 3

8 0 4

7 6 5

g: 当前深度：5

h: 启发函数：0

f=g+h= 5

open表长6

分析：在本次实验选用的数据中，采用不在位数和哈密顿距离两种启发函数所得到的结果相同。但是不同的启发函数在解题过程中其实是会对结果产生较大影响的，估价函数由位深加上启发函数得到，每次扩展open表中状态的顺序由估价函数大小决定，所以不同的启发函数会影响每次扩展的状态选择，从而会在整个过程中产生较大的区别。

#### 五 根据宽度优先搜索算法和A`算法求解8数码问题的结果,分析启发式搜索的特点。

请输入初始状态...

2 8 3 1 6 4 7 0 5

请输入目标状态...

1 2 3 8 0 4 7 6 5

--------------------

起始状态逆序数为 11 为奇

目标状态逆序数为 7 为奇

至少要移动37步!

2 8 3

1 6 4

7 0 5

g: 当前深度：0

h: 启发函数：0

f=g+h=0

2 8 3

1 0 4

7 6 5

g: 当前深度：1

h: 启发函数：0

f=g+h=1

2 8 3

1 6 4

0 7 5

g: 当前深度：1

h: 启发函数：0

f=g+h=1

2 8 3

1 6 4

7 5 0

g: 当前深度：1

h: 启发函数：0

f=g+h=1

2 0 3

1 8 4

7 6 5

g: 当前深度：2

h: 启发函数：0

f=g+h=2

2 8 3

0 6 4

1 7 5

g: 当前深度：2

h: 启发函数：0

f=g+h=2

2 8 3

1 6 0

7 5 4

g: 当前深度：2

h: 启发函数：0

f=g+h=2

2 8 3

0 1 4

7 6 5

g: 当前深度：2

h: 启发函数：0

f=g+h=2

2 8 3

1 4 0

7 6 5

g: 当前深度：2

h: 启发函数：0

f=g+h=2

2 3 0

1 8 4

7 6 5

g: 当前深度：3

h: 启发函数：0

f=g+h=3

2 8 0

1 6 3

7 5 4

g: 当前深度：3

h: 启发函数：0

f=g+h=3

2 8 3

7 1 4

0 6 5

g: 当前深度：3

h: 启发函数：0

f=g+h=3

2 8 3

1 4 5

7 6 0

g: 当前深度：3

h: 启发函数：0

f=g+h=3

0 8 3

2 6 4

1 7 5

g: 当前深度：3

h: 启发函数：0

f=g+h=3

0 2 3

1 8 4

7 6 5

g: 当前深度：3

h: 启发函数：0

f=g+h=3

2 8 3

1 0 6

7 5 4

g: 当前深度：3

h: 启发函数：0

f=g+h=3

2 8 3

6 0 4

1 7 5

g: 当前深度：3

h: 启发函数：0

f=g+h=3

2 8 0

1 4 3

7 6 5

g: 当前深度：3

h: 启发函数：0

f=g+h=3

0 8 3

2 1 4

7 6 5

g: 当前深度：3

h: 启发函数：0

f=g+h=3

2 3 4

1 8 0

7 6 5

g: 当前深度：4

h: 启发函数：0

f=g+h=4

2 8 3

7 1 4

6 0 5

g: 当前深度：4

h: 启发函数：0

f=g+h=4

2 0 8

1 4 3

7 6 5

g: 当前深度：4

h: 启发函数：0

f=g+h=4

8 0 3

2 1 4

7 6 5

g: 当前深度：4

h: 启发函数：0

f=g+h=4

2 0 8

1 6 3

7 5 4

g: 当前深度：4

h: 启发函数：0

f=g+h=4

2 8 3

6 7 4

1 0 5

g: 当前深度：4

h: 启发函数：0

f=g+h=4

2 0 3

6 8 4

1 7 5

g: 当前深度：4

h: 启发函数：0

f=g+h=4

2 8 3

1 4 5

7 0 6

g: 当前深度：4

h: 启发函数：0

f=g+h=4

2 8 3

0 1 6

7 5 4

g: 当前深度：4

h: 启发函数：0

f=g+h=4

2 8 3

6 4 0

1 7 5

g: 当前深度：4

h: 启发函数：0

f=g+h=4

2 0 3

1 8 6

7 5 4

g: 当前深度：4

h: 启发函数：0

f=g+h=4

8 0 3

2 6 4

1 7 5

g: 当前深度：4

h: 启发函数：0

f=g+h=4

2 8 3

1 5 6

7 0 4

g: 当前深度：4

h: 启发函数：0

f=g+h=4

1 2 3

0 8 4

7 6 5

g: 当前深度：4

h: 启发函数：0

f=g+h=4

2 3 4

1 0 8

7 6 5

g: 当前深度：5

h: 启发函数：0

f=g+h=5

2 8 3

7 0 4

6 1 5

g: 当前深度：5

h: 启发函数：0

f=g+h=5

2 6 8

1 0 3

7 5 4

g: 当前深度：5

h: 启发函数：0

f=g+h=5

2 8 3

1 5 6

7 4 0

g: 当前深度：5

h: 启发函数：0

f=g+h=5

1 2 3

8 0 4

7 6 5

g: 当前深度：5

h: 启发函数：0

f=g+h=5

分析：宽度优先算法也就是启发式搜索算法的启发函数h(n)为0时的搜索，是一种高价搜索。它遍历所有的可能性，扩展出每一种可能的状态，这样会消耗大量的计算资源，虽说一定可以找到解，但在求解复杂问题时并不可取。在本实验中，我们可以看出，启发式搜索的速度、效率明显高于宽度优先搜索，启发式搜索只需要进行6次迭代而宽度搜索需要38次。

由此看来，启发式搜索能够利用与该问题有关的信息来简化搜索过程，可以节省算力，使计算机更快得出结果。但是启发式算法也是极容易出错的，一个启发式搜索可能得到一个次优解，也可能一无所获。

#### 六 代码

#include<iostream>

#include<cmath>

#include "stdlib.h"

#include "time.h"

#include<stack>

#include<queue>

#include<vector>

using namespace std;

#define num 9

class Node {

private:

int depth = 0;//深度

int dislocation;//不在位数

int value;//估价值

int hamilton;//哈密顿距离

public:

int state[9];

//用于设置私有成员

void setDepth(int x) { depth = x; }

void setDislocation(int x) { dislocation = x; }

void setValue(int x) { value = x; }

void setHamilton(int x) { hamilton = x; }

//接口函数，用于调用私有成员

int depth\_() { return depth; }

int dislocation\_() { return dislocation; }

int value\_() { return value; }

int hamilton\_() { return hamilton; }

//按照估价值小的方案构造优先级队列，若估价值相等，则按照不在位数小构造

friend bool operator < (Node A, Node B) {

if (A.value == B.value) return A.dislocation > B.dislocation;

return A.value > B.value;

}

//重载==运算符，定义结点相等为每位都相等

bool operator == (Node A) {

for (int i = 0; i < num - 1; i++) {

if (state[i] != A.state[i]) return false;

}

return true;

}

//计算不在位数

int cal\_Dislocation(Node A) {

int d = 8;

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

if (state[i] == A.state[i] && A.state[i] != 0) d--;

}

//return d;

return 0;

}

//利用不在位数计算估价值

int cal\_value(Node T) {

int n = 8;

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

if (state[i] == T.state[i] && T.state[i] != 0) n--;

}

//return n + depth;

return n;

}

//计算哈密顿距离

int cal\_Hamilton(Node T) {

int n = 0;

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

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

if (state[i] == 0) continue;

else if (state[i] != T.state[j]) continue;

else n = n + abs((i / 3) - (j / 3)) + abs((i % 3) - (j % 3));

}

}

//return n;

return 0;

}

//利用哈密顿距离计算估价值

int cal\_value\_1(Node T) {

int n = 0;

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

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

if (state[i] == 0) continue;

else if (state[i] != T.state[j]) continue;

else n = n + abs((i / 3) - (j / 3)) + abs((i % 3) - (j % 3));

}

}

//return n + depth;

return depth;

}

};

priority\_queue<Node> open;//open表，存放带扩展节点；优先队列priority\_queue，可自动对加入的对象排序

vector<Node> close;//close表，存放已扩展节点

//逆序数奇偶性检查

int check\_nixu(Node& Ini, Node& Tar) {

//计算始末状态的逆序数

int nixu\_i = 0, nixu\_t = 0;

for (int i = 0; i <= num - 2; i++) {

for (int j = i + 1; j < num; j++) {

if (Ini.state[i] > Ini.state[j] && Ini.state[i] \* Ini.state[j] != 0)

nixu\_i++;

}

}

for (int i = 0; i <= num - 2; i++) {

for (int j = i + 1; j < num; j++)

if (Tar.state[i] > Tar.state[j] && Tar.state[i] \* Tar.state[j] != 0)

nixu\_t++;

}

//输出始末状态逆序数

if (nixu\_i % 2 == 1) cout << "起始状态逆序数为 " << nixu\_i << " 为奇" << endl;

if (nixu\_i % 2 == 0) cout << "起始状态逆序数为 " << nixu\_i << " 为偶" << endl;

if (nixu\_i % 2 == 1) cout << "目标状态逆序数为 " << nixu\_t << " 为奇" << endl;

if (nixu\_i % 2 == 0) cout << "目标状态逆序数为 " << nixu\_t << " 为偶" << endl;

//判断逆序数奇偶性是否相同，相同才有解

if (nixu\_i % 2 != nixu\_t % 2){

return 0;

}

return 1;

}

//输入格式检查

int check\_in(Node A, Node B) {

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

if (A.state[i] >= 0 && A.state[i] <= 8 && B.state[i] >= 0 && B.state[i] <= 8) continue;

else return 0;

}

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

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

if (i == j)continue;

else if (A.state[i] == A.state[j] || B.state[i] == B.state[j]) return 0;

}

}

return 1;

}

//用于判断两结点是否相同

int judge(Node A, Node B)

{

for (int i = 0; i <= 8; i++){

if (A.state[i] != B.state[i]){

return 0;

}

}

return 1;

}

//产生新节点，加入OPEN表

void creatNode(Node& A, Node B)

{

//确定空格位置后，上下左右换位形成子节点，排除超出范围的换位操作

int blank;

for (blank = 0; blank < 9 && A.state[blank] != 0; blank++);//确定空格在一维数组中的序号

int x = blank / 3, y = blank % 3; //空格对应到3\*3网格中的坐标

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

{

int new\_x = x, new\_y = y;

int flag = 0;

Node temp;

//分别朝四个方向移动空格

if (d == 0) new\_x = x - 1;

if (d == 1) new\_y = y - 1;

if (d == 2) new\_x = x + 1;

if (d == 3) new\_y = y + 1;

//用于记录移动后空格的位置

int new\_blank = new\_x \* 3 + new\_y;

//若移动过后仍在3\*3范围内，继续进行操作

if (new\_x >= 0 && new\_x < 3 && new\_y >= 0 && new\_y < 3){

//temp用于存放新产生的结点

temp = A;

temp.state[blank] = A.state[new\_blank];

temp.state[new\_blank] = 0;

//检查新产生的节点是否在close表中

for (auto it = close.begin(); it != close.end(); ++it) {

if (\*it == temp) { flag=1; continue; }

}

if (flag == 0) {

//利用不在位数计算

//temp.setDislocation(temp.cal\_Dislocation(B));

//利用哈密顿距离计算

temp.setHamilton(temp.cal\_Hamilton(B));

temp.setValue(temp.cal\_value\_1(B));//可选择不同的估价值计算方式，计算新结点的估价值

temp.setDepth(A.depth\_() + 1);//新结点的深度为原节点+

open.push(temp);//新结点进入open表

}

}

}

}

int main(){

Node Ini, Tar;

cout << "请输入初始状态..."<<endl;

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

cin >> Ini.state[i];

cout << "请输入目标状态..."<<endl;

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

cin >> Tar.state[i];

cout << "--------------------" << endl;

if (!check\_in(Ini, Tar)) {

cout << "输入错误！" << endl;

return 0;

}

if (!check\_nixu(Ini, Tar)){

cout << "始末状态逆序数奇偶性不同，无解!" << endl;

return 0;

}

open.push(Ini);//初始结点进入open表

int max = 20000;//最多循环20000次

int num\_ = 0;

while (1){

//将open表中优先级最高的元素尾插入close表中，并在open表中删除该结点

close.push\_back(open.top());

open.pop();

//对比当前状态与目标状态，不相同则拓展当前结点，否则退出循环

if (!judge(close.back(), Tar)) {

creatNode(close.back(), Tar);

}

else break;

// cout << max-- << endl;

if (max == 0) {

cout << "搜索失败！" << endl;

break;

}

}

// 打印移动过程

cout << "至少要移动" << close.size() - 1 << "步!" << endl;//搜索步数为close表中元素个数-1

for (auto it = close.begin(); it != close.end(); ++it) {

for (int i = 0; i <= 8; i++) {

cout << (\*it).state[i] << " ";

if ((i + 1) % 3 == 0) cout << endl;

}

//利用不在位数计算

(\*it).setDislocation((\*it).cal\_Dislocation(Tar));

cout << "g: 当前深度：" << (\*it).depth\_() << endl;

cout << "h: 启发函数：" << (\*it).dislocation\_() << endl;

cout << "f=g+h=" << (\*it).depth\_() + (\*it).dislocation\_() << endl;

cout << "\n";

//利用哈密顿距离计算

/\*(\*it).setHamilton((\*it).cal\_Hamilton(Tar));

cout << "g: 当前深度：" << (\*it).depth\_() << endl;

cout << "h: 启发函数：" << (\*it).hamilton\_() << endl;

cout << "f=g+h= " << (\*it).depth\_() + (\*it).hamilton\_() << endl;

cout << "open表长" << open.size() << endl;

cout << "\n";\*/

}

return 0;

}

#### 七 心得体会

在此之前，我对各类搜索策略的了解仅仅只停留在课堂上的理论学习部分，通过本次实验，我加深了对各类搜索策略的理解，也明确了各类搜索策略的优缺点。在实验前研究例程时，并不知道程序有bug，当发现程序的输出与我对课本的理解不相符时，我多次翻看课本，在这个过程中，我对第五章的知识有了更深入的了解。