装

订

线

同济大学

人工智能实验报告

（八数码）

学　号　：1453381

姓　名　：曾 鸣

班　级　：计算机二班

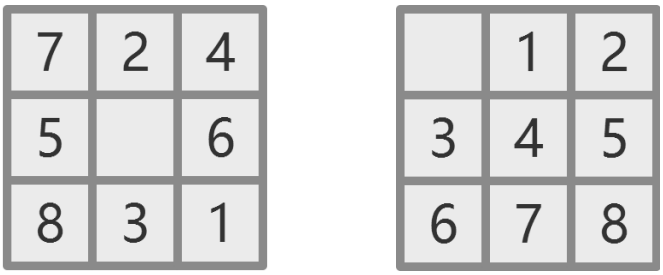
日　期　：2017 年 4 月 16 日

1. 题目：八数码

1.1 题目描述

在 3×3 方格盘上，放有八个数码，剩下一个位置为空，每一空格其上下左右的数码可移至空格。给定初始位置和目标位置，要求通过一系列的数码移动，将初始状态转化为目标状态。状态转换的规则：空格四周的数移向空格，我们可以看作是空格移动，它最多可以有 4 个方向的移动，即上、下、左、右。九宫重排问题的求解方法，就是从给定的初始状态出发，不断地空格上下左右的数码移至空格，将一个状态转化成其它状态，直到产生目标状态。

例如：



初始状态　　目标状态

八数码问题示意图

1.2 形式化定义

- **状态：**状态指定了 8 个棋子中的每一个以及空位在棋盘的 9 个方格上的分步。
- **初始状态：**任何状态都可以被指定为初始状态。注意要到达任何一个给定的目标，可能的初始状态中恰好只有一半可以作为开始。
- **后继函数：**用来产生通过四个行动（把空位向 Left，Right，Up 或 Down 移动）能够达到的合法状态。
- **目标测试：**用来检测状态是否能匹配到目标布局。
- **路径耗散：**每一个耗散值为 1，因此整个路径的耗散值是路径中的步数。

2. 实验目的和环境

2.1 实验目的

熟悉人工智能系统中的问题求解过程；
熟悉状态空间的启发式搜索算法的应用；
熟悉对八数码问题的建模、求解及编程语言的应用。

2.2 实验环境

硬件环境：

计算机型号：惠普 Pavilion M4

内存：4.00GB

CPU：Intel Core i5　2.6GHz

软件环境：

操作系统：Windows10 版本

IDE：Dev-C++ 5.10 版

实现语言：C++（C++11 标准）

3. 算法和实现

3.1 A*算法介绍

A*算法是一种常用的启发式搜索算法。

在 A*算法中，一个结点位置的好坏用估价函数来对它进行评估。A*算法的估价函数可表示为：

f'(n) = g'(n) + h'(n)

f'(n)是估价函数；

g'(n)是起点到终点的最短路径值（也称为最小耗费或最小代价）；

h'(n)是 n 到目标的最短路经的启发值；

由于 f'(n)其实是无法预先知道的，所以实际上使用的是下面的估价函数：

f(n) = g(n) + h(n)

g(n)是从初始结点到节点 n 的实际代价；

h(n)是从结点 n 到目标结点的最佳路径的估计代价；

在这里主要是 h(n)体现了搜索的启发信息，因为 g(n)是已知的。用 f(n)作为 f'(n)的近似，也就是用 g(n)代替 g'(n)，h(n)代替 h'(n)。这样必须满足两个条件：

（1）g(n)>=g'(n)（大多数情况下都是满足的，可以不用考虑），且 f 必须保持单调递增。

（2）h 必须小于等于实际的从当前节点到达目标节点的最小耗费 h(n)<=h'(n)。

第二点特别的重要。可以证明应用这样的估价函数是可以找到最短路径的。

本次实验相应的具体估价函数、代价函数、启发值函数：

g(n)：本次试验中该状态的耗费值自然是从初始状态到达该状态的最少步数；

h(n)：启发值可以选用不再目标状态的数码个数，也可以选用曼哈顿距离（及当前状态中每个数码到达相应位置所需最少步数），但是曼哈顿距离更适合，所以选用曼哈顿距离作为启发值。

0 1 2 1 2 3 2 3 4
1 0 1 2 1 2 3 2 3
2 1 0 1 2 1 2 3 2
1 2 1 0 1 2 1 2 3
2 1 2 1 0 1 2 1 2
3 2 1 2 1 0 1 2 1
2 3 2 1 2 1 0 1 2
3 2 3 2 1 2 1 0 1
4 3 2 3 2 1 2 1 0

上述矩阵第 i 行第 j 列表示数字 i-1 在八数码中第 j 位置的曼哈顿距离。

3.2 算法伪代码

算法的功能：产生 8 数码问题的解(由初始状态到达目标状态的过程）

输入：初始状态，目标状态

输出：从初始状态到目标状态的一系列过程

算法描述：

Begin：

根据八数码奇偶排列的性质，判断初始状态是否可解；

If(问题可解)

计算初始状态评价函数值 f，g，h；

将初始状态加入优先队列 Q，该节点的父节点为空；

While（优先队列非空）

① 将优先队列 Q 中评价值最小的节点 qn 作为当前结点并出队；

② 如果 qn 节点状态已经扩展过，跳转到①；

③ 判断当前结点状态和目标状态是否一致，若一致，跳出循环；

④ 对当前结点，分别按照上、下、左、右方向移动空格位置来扩展新的状态结点，计算新扩展结点的评价值 f，g，h，记录其父节点为该节点，将新扩展节点加入 Q；

End while

End if

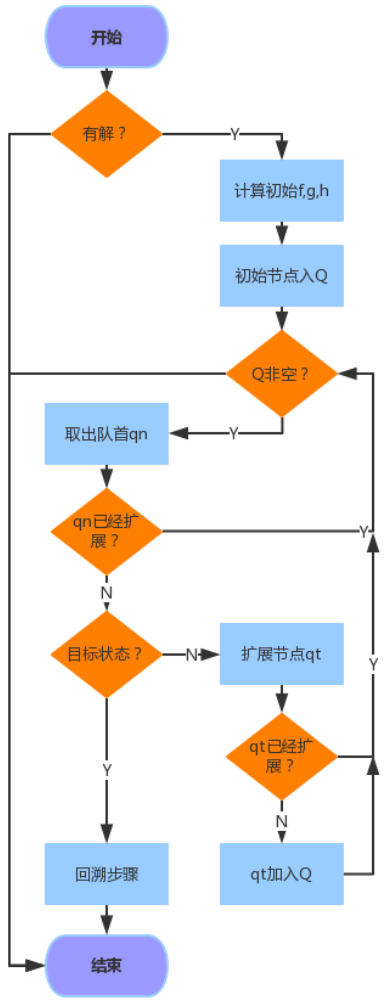
从目标状态依次寻找父节点直到初始状态，输出结果；

End

注：

算法中所用的**优先队列**是一种数据结构（堆结构）。按照 A*算法需要每次对队列中的待扩展节点进行排序，如果有 n 个节点，则每次排序最快时间复杂度为 O(nlogn)，这样效率较低。事实上，我们只需每次得到待扩展队列中 fn 值最小元素，小顶堆式的优先队列的顶端是 fn 最小的节点，每次插入或提取优先队列中的一个节点的时间复杂度为 O(logn)，效率得到提升。

3.3 算法流程图



3.4 实现优化

7	2	4
5		6
8	3	1

初始状态

	1	2
3	4	5
6	7	8

目标状态

在具体记录每个状态时，我们可以用一维数组或者二维数组记录相应位置的数字，但是这样效率不高，我们可以用一个 32 位整型数来表示每一个状态。因为数字只有 8 个，可以用 3 个 bit 完全表示，最高位的 32-3*9=5 个 bit 来记录空格位置。这样算法的效率可以提升一个数量级。

初始状态用 int 数来表示时具体描述：

pos0	8	7	6	5	4	3	2	1	0
00100	000	010	111	101	000	100	011	001	110

如上：

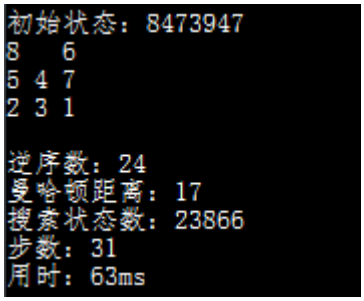
初始状态为：0x205E88CE（0010 0000 0101 1110 1000 1000 1100 1110）

目标状态为：0x07D63440（0000 0111 1101 0110 0011 0100 0100 0000）

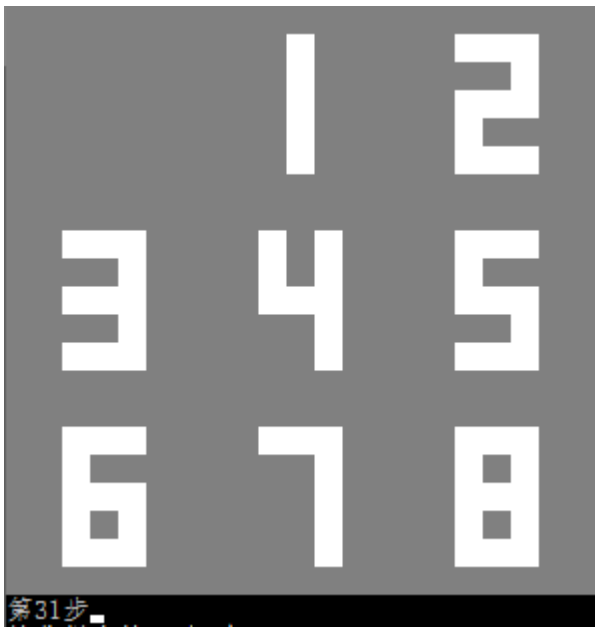
4. 结果验证

用逆向广度优先搜索可以知道，八数码最多解的步数不会超过 31 步，选取最多步数的解来进行验证，如下截图：

求解记录：



效果展示：



装 订 线

文件中求解过程：
解的步数为：31

第 0 步：
8 6
5 4 7
2 3 1

第 1 步：
8 6
5 4 7
2 3 1

第 2 步：
5 8 6
4 7
2 3 1

第 3 步：
5 8 6
2 4 7
3 1

第 4 步：
5 8 6
2 4 7
3 1

第 5 步：
5 8 6
2 4 7
3 1

第 6 步：
5 8 6
2 4
3 1 7

第 7 步：
5 8
2 4 6
3 1 7

第 8 步：
5 8
2 4 6
3 1 7

第 9 步：
5 4 8
2 6
3 1 7

第 10 步：
5 4 8
2 1 6
3 7

第 11 步：
5 4 8
2 1 6
3 7

第 12 步：
5 4 8
2 1
3 7 6

第 13 步：
5 4
2 1 8
3 7 6

第 14 步：
5 4
2 1 8
3 7 6

第 15 步：
5 4
2 1 8
3 7 6

第 16 步：
2 5 4
1 8
3 7 6

第 17 步：
2 5 4
1 8
3 7 6

第 18 步：
2 5 4
1 7 8
3 6

第 19 步：
2 5 4
1 7 8
3 6

第 20 步：
2 5 4
1 7
3 6 8

第 21 步：
2 5
1 7 4
3 6 8

第 22 步：
2 5
1 7 4
3 6 8

第 23 步：
2 5
1 7 4
3 6 8

第 24 步：
1 2 5
7 4
3 6 8

第 25 步：
1 2 5
3 7 4
6 8

第 26 步：
1 2 5
3 7 4
6 8

第 27 步：
1 2 5
3 4
6 7 8

第 28 步：
1 2 5
3 4
6 7 8

第 29 步：
1 2
3 4 5
6 7 8

第 30 步：
1 2
3 4 5
6 7 8

第 31 步：
1 2
3 4 5
6 7 8

装 订 线

无解验证：
当输入为非法输入或者不能到达目标状态时不会有解：

5. 附件：源程序

```
//core.cpp
#define _CRT_SECURE_NO_WARNINGS
#include<cmath>
#include<ctime>
#include<cstdlib>
#include<iostream>
#include<fstream>
#include<vector>
#include<queue>
#include<unordered_map>
#include<conio.h>
#include<Windows.h>
using namespace std;

//宏定义终止状态
#define endState 0x07D63440

//mhd[9][9]记录每个数字不同位置的曼哈顿距离
int mhd[9][9];

//计算曼哈顿距离
void calcuMhd() {
    int pos, n;
    for (n = 0; n < 9; ++n)
        for (pos = 0; pos < 9; ++pos) {
            int d = abs(n - pos);
            mhd[n][pos] = d / 3 + d % 3;
        }
}

//测试方式一：随机化产生初始序列
void generInitState(unsigned int &initState) {
    srand((unsigned)time(NULL));
    initState = 0;
    unsigned int use = 0, pos0 = 0;
    int i;
    for (initState = 0, i = 0; i < 9;) {
```

```
            unsigned int r = (unsigned)rand() % 9;
            if (!(use & 1 << r)) {
                ++i;
                use |= 1 << r;
                if (r == 0)
                    pos0 = 9 - i;
                else
                    --r;
                initState = initState << 3 | r;
            }
        }
        initState |= pos0 << 27;
    }

    //测试方式二：输入初始状态
    void inputState(unsigned int &initState) {
        int i, n,pos0;
        initState = 0;
        for (i = 0; i <=8; ++i) {
            cin >> n;
            if (n)
                --n;
            else
                pos0 = i;
            initState |= n << 3 * i;
        }
        initState |= pos0 << 27;
    }

    //计算始末状态逆序数
    int reOrderNum(unsigned int state,int flag=1) {
        int n = 0;
        unsigned int i, j, t1, pos0;
        for (pos0 = state >> 27, i = 0; i < 27; i += 3)
            if (pos0 != i / 3)
                for (t1 = state >> i & 7, j = 0; j < i; j += 3)
                    if (t1 < (state >> j & 7))
                        ++n;
        if (flag)cout << dec << "逆序数：" << n << endl;
        return n;
    }

    //判断初始状态到目标状态是否可解
    bool canSolving(unsigned int initState) {
        return (reOrderNum(initState) & 1) == (reOrderNum(endState,0) & 1);
    }

    //回溯搜索解的步骤
    void findStep(vector<int>&step, unordered_map<int, int>&hashMap, int state) {
        int preState = hashMap[state];
        if (!preState) {
            step.push_back(state);
            return;
        }
```

```
    findStep(step, hashMap, preState);
    step.push_back(state);
}

//搜索中的节点信息
struct queueNode {
    int state;        //状态
    int f;            //评估值fn
    int g;            //耗散值gn
    int h;            //启发值hn
    int fa;           //记录扩展到该状态的父状态

    //小顶堆的比较函数，堆顶为fn最小
    bool operator < (const queueNode& thn)const {
        return f > thn.f;
    }
};

//输出一个状态到窗口或文件
void outState(ostream &fout, int s) {
    int pos0, n1, j;
    for (pos0 = s >> 27, j = 0; j < 9; ++j) {
        n1 = s >> j * 3 & 7;
        if (j == pos0)
            fout << "  ";
        else
            fout << n1 + 1 << " ";
        if (j % 3 == 2)
            fout << endl;
    }
    fout << endl;
}

//输出解的步骤到文件
void outPutFile(vector<int>&step) {
    ofstream fout("result.txt", ios::out);
    fout << "解的步数为：" << step.size() - 1 << endl;
    for (int i = 0; i < (int)step.size(); ++i) {
        fout << "第" << i << "步：" << endl;
        outState(fout, step[i]);
        fout << endl;
    }
}

//A*算法实现
void aStar(const int initState, vector<int>&step) {
    unordered_map<int, int>hashMap;          //记录已确定状态
    priority_queue<queueNode>Q;              //优先队列小顶堆保存待扩展节点状态
    int dir[4] = { -1,1,-3,3 };

    //初始化初始节点
    queueNode qn = { initState,0,0,0,0 };
    int i,initMhd,n1, cut, pos0, npos0, nextState;
    for (pos0 = initState >> 27, i = 0; i < 27; i += 3)
```

```
        if (i / 3 != pos0) {
            n1 = initState >> i & 7;
            qn.h += mhd[n1 + 1][i / 3];
        }
    initMhd=qn.f = qn.g + qn.h;
    Q.push(qn);

    //开始搜索
    while (!Q.empty()) {
        qn = Q.top();//取fn值最小的状态节点
        Q.pop();

        //该节点状态未确定
        if (hashMap.count(qn.state) == 0) {
            hashMap[qn.state] = qn.fa;      //哈希表保存其父状态
            if (qn.state == endState)       //搜索到目标状态
                break;

            //该节点四个可能的方向进行扩展
            for (pos0 = qn.state >> 27, i = 0; i < 4; ++i) {
                npos0 = pos0 + dir[i];

                //判断是否可以扩展
                if (npos0 >= 0 && npos0 <= 8 && (npos0 / 3 == pos0 / 3 || npos0 % 3 == pos0 %
3)) {
                    n1 = qn.state >> npos0 * 3 & 7;
                    nextState = (qn.state^n1 << npos0 * 3) | n1 << pos0 * 3;
                    nextState ^= pos0 << 27;
                    nextState |= npos0 << 27;

                    if (hashMap.count(nextState))  //扩展的新状态未确定
                        continue;
                    cut = mhd[n1 + 1][npos0] - mhd[n1 + 1][pos0];
                    Q.push({ nextState,qn.g + 1 + qn.h - cut,qn.g + 1,qn.h - cut,qn.state });
                }
            }
        }
    }
    cout << "曼哈顿距离：" << initMhd << endl;
    cout << "搜索状态数：" << dec << hashMap.size() << endl;
    findStep(step, hashMap, endState);
}

bool digit8(vector<int>&step, unsigned int initState) {
    bool suc;
    cout << "初始状态：" << hex << initState << endl;
    outState(cout, initState);
    if (!(suc = canSolving(initState)))
        cout << "初始状态无解" << endl;
    else {
        aStar(initState, step);
        cout << "步数：" << step.size() - 1 << endl;
        outPutFile(step);
    }
```

装 订 线

```
    return suc;
}

//调用display.cpp中的演示函数
void display(const vector<int>&);

int main() {
    freopen("out.txt", "w", stdout);
    vector<int>step;
    unsigned int initState = 0;
    calcuMhd();

    //三种方式进行输入初始状态,可任选其一
    inputState(initState);     //手动输入
//  generInitState(initState); //随机产生
                               //赋值方式
//  initState=0x85DD981;       //2 0 7 5 6 4 8 3 1        无解
//  initState=0x205E88CE;      //7 2 4 5 0 6 8 3 1        书上26步解
//  initState = 0x08473947;    //8 0 6 5 4 7 2 3 1        31步解
    double start = GetTickCount();
    bool suc = digit8(step, initState);
    double end = GetTickCount();
    cout << "用时：" << end - start << "ms" << endl;

    //成功有解则演示
    if (suc) {
        cout << endl << "按任意键演示";
        getch();
        display(step);
    }
    return 0;
}
```

装 订 线

装 订 线

```
//display.cpp
#define _CRT_SECURE_NO_WARNINGS
#include<Windows.h>
#include<conio.h>
#include<iostream>
#include<vector>
using namespace std;
#define FRONT 15
#define BACK 8

typedef BOOL(WINAPI *PROCSETCONSOLEFONT)(HANDLE, DWORD);

void setcolor(HANDLE hout, const int bg_color, const int fg_color)
{
    SetConsoleTextAttribute(hout, bg_color * 16 + fg_color);
}
void setconsolefont(const HANDLE hout, const int font_no)
{
    HMODULE hKernel32 = GetModuleHandleA("kernel32");
    PROCSETCONSOLEFONT setConsoleFont = (PROCSETCONSOLEFONT)GetProcAddress(hKernel32,
"SetConsoleFont");
    setConsoleFont(hout, font_no);
    return;
}
void gotoxy(HANDLE hout, const int X, const int Y)
{
    COORD coord;
    coord.X = X;
    coord.Y = Y;
    SetConsoleCursorPosition(hout, coord);
}

void setconsoleborder(const HANDLE hout, const int cols, const int lines)
{
    char cmd[80];
    system("cls");
    sprintf(cmd, "mode con cols=%d lines=%d", cols, lines);
    system(cmd);
    return;
}

int num[9][7] = {
    { 0,0,0,0,0,0,0 },
    { 0,0xC0,0xC0,0xC0,0xC0,0xC0,0 },
    { 0,0x3F0,0x30,0x3F0,0x300,0x3F0,0 },
    { 0,0x3F0,0x30,0x3F0,0x30,0x3F0,0 },
    { 0,0x330,0x330,0x3F0,0x30,0x30,0 },
    { 0,0x3F0,0x300,0x3F0,0x30,0x3F0,0 },
    { 0,0x3F0,0x300,0x3F0,0x330,0x3F0,0 },
    { 0,0x3F0,0x30,0x30,0x30,0x30,0 },
    { 0,0x3F0,0x330,0x3F0,0x330,0x3F0,0 }
};

void displayOneState(int state, HANDLE hout) {
```

```
    int col, row, i, j, pos0, pos1, n;
    pos0 = state >> 27;
    for (row = 0; row < 3; ++row)
        for (col = 0; col < 3; ++col) {
            n = state >> (pos1 = row * 3 + col) * 3 & 7;
            pos0 == pos1 ? n : ++n;
            for (i = 0; i < 7; ++i) {
                for (j = 0; j < 14; ++j) {
                    gotoxy(hout, col * 14 + j, row * 7 + i);
                    if (num[n][i] >> (13 - j) & 1)
                        setcolor(hout, FRONT, FRONT);
                    else
                        setcolor(hout, BACK, BACK);
                    printf(" ");
                }
                printf("\n");
            }
        }
    Sleep(50);
    //_getch();
}

void display(const vector<int>&step) {
    const HANDLE hout = GetStdHandle(STD_OUTPUT_HANDLE);
    setconsoleborder(hout, 46, 23);
    setconsolefont(hout, 0);
    for (int i = 0; i < (int)step.size(); ++i) {
        displayOneState(step[i], hout);
        setcolor(hout, 0, 7);
        printf("第%d步", i);
    }
    _getch();
}
```

装 订 线