重庆交通大学计算机与信息学院

验 证 性 实 验 报 告

班　　级：计科　专业 14 级 1 班

学　　号：631406010109

姓　　名：郭文浩

实验项目名称：状态空间搜索

实验项目性质：验证性实验

实验所属课程：人工智能

实验室(中心)：软件中心实验室（语音楼 8 楼）

指 导 教 师 ：朱振国

实验完成时间：2017 年 5 月 18 日

评阅意见：
实验成绩：　　签名：　　年　月　日

一、实验目的

理解和掌握状态空间搜索的策略。

二、实验内容及要求

㈠ 实验内容：在一个 3*3 的九宫中有 1-8 个数码及一个空格随即的摆放在其中的格子里，现在要求实验这个问题：将该九宫格调整为某种有序的形式。调整的规则是，每次只能将与空格（上、下、左、右）相邻的一个数字平移到空格中。

㈡ 实验要求：用选定的编程语言编写程序，利用不同的搜索策略进行状态空间搜索（如宽度优先搜索、深度优先搜索、有界深度优先搜索等）。

三、实验设备及软件

㈠ 实验设备：重庆交通大学语音楼八楼机房的计算机和自己的笔记本。

㈡ 实验软件：eclipse。

四、设计方案

㈠ 题目

利用状态空间搜索算法解决八数码问题

㈡ 设计的主要思路

八数码问题，是对给定的一个初始位置，然后经过多次移动找到目标位置，并列举出其中的移动过程，最后可以找到既是可以成功，否则以失败告终。是一种过程中无人为参与的一中求解方法。

深度优先搜索：

深度优先搜索算法是按照一条路径一直往下深度延伸其子节点的算法，直到找到答案为止。也可能一直到达一个很深的深度之后还是没有找到问题的答案，这样就有可能出现栈溢出或者内存超界的情况（本实验中数字组合相对较少，不会造成内存超限的情况）。在深度优先搜索的求解过程中使用栈来存储还未搜索的节点，已

经搜索过的节点使用一个链表来存储（避免重复的搜索，属于优化过程）。如果已经在链表中那么就不在放到栈中，因为之前已经搜索过了，不需要重复搜索。当然在深度优先搜索的过程中，不需要记录其深度，因为不会用深度来限制搜索。所以判断是不是已经包含当然搜索的节点在链表中，是用其中的数值数组是不是完全相同来判断的。如果找不到就返回“深度优先搜索失败”。

有界深度优先搜索：

有界深度优先搜索是在深度优先搜索的基础上进行的另一种对其深度进行限制的一种搜索方法。当然其使用的数据结构也是和深度优先搜索一样的，其中的不同之处在于，在有界深度优先搜索的基础上判断是不是相等时会有一个深度的同时相等的判断，即只有当其中的数组数值和搜索深度同时相等时才认为是相等的。同时也会在深度达到 5 以后搜索不会继续发展更深的子节点，而是开始搜索其兄弟节点等，然后继续。如果找不到就返回“有限深度优先搜索失败”。

广度（宽度）优先搜索：

广度优先搜索，也被称作宽度优先搜索，是首先在兄弟节点之间进行搜索和遍历的方法，然后知道其全部的兄弟节点都已经遍历完，才开始继续往下发展其子节点，所以在这个过程中使用队列（Queue 接口的实现类 LinkedList）来存储其中还没有遍历的但是已经发展了的节点，同时使用链表来存储已经搜索过的子节点，然后基本上是和深度优先搜索算法类似。如果找不到就返回“广度优先搜索失败”。

㈢ 主要功能

输入初始位置和目标位置，再输入有限深度搜索的最大界限，程序会自动运行深度、有限深度和广度三个算法，并展示中间的转换步骤。

五、主要代码

```
// 深度优先搜索
public void depthFirstSearch() {
    System.out.println("\n现在开始深度优先搜索方法路径！");
    if (!searched_list.isEmpty())
        searched_list.clear();
    while (!ep_stack.isEmpty()) {
        EightPuzzle move_ep = ep_stack.pop();
        depth = move_ep.getDepth();
        move_ep.getPostion();
        int x = move_ep.getBlankPos_x(), y = move_ep.getBlankPos_y();
        move_ep.print();
        searched_list.add(move_ep);
        if (move_ep.isEquals(target_ep)) {
```

```
            System.out.println("深度优先搜索成功！");
            return;
        }
        depth++;
        EightPuzzle temp = null;

        temp = move_ep.depthClone();

        if (EightPuzzleOperator.canMove(x, y, 1)) {
            temp = EightPuzzleOperator.movePosition(move_ep, 1);
            temp.setDepth(depth);
            if (!searched_list.contains(temp)) {
                ep_stack.push(temp);
            }
        }
        if (EightPuzzleOperator.canMove(x, y, 2)) {
            temp = EightPuzzleOperator.movePosition(move_ep, 2);
            temp.setDepth(depth);
            if (!searched_list.contains(temp)) {
                ep_stack.push(temp);
            }
        }
        if (EightPuzzleOperator.canMove(x, y, -1)) {
            temp = EightPuzzleOperator.movePosition(move_ep, -1);
            temp.setDepth(depth);
            if (!searched_list.contains(temp)) {
                ep_stack.push(temp);
            }
        }
        if (EightPuzzleOperator.canMove(x, y, -2)) {
            temp = EightPuzzleOperator.movePosition(move_ep, -2);
            temp.setDepth(depth);
            if (!searched_list.contains(temp)) {
                ep_stack.push(temp);
            }
        }
    }
    System.out.println("深度优先搜索失败！");
}

// 深度优先搜索(有界，最大深度为maxDepth)
public void boundedDepthFirstSearch() {
```

```
        System.out.println("\n现在开始有界深度优先搜索方法路径！");
        if (!searched_list.isEmpty())
            searched_list.clear();
        while (!ep_stack.isEmpty()) {
            EightPuzzle move_ep = ep_stack.pop();
            depth = move_ep.getDepth();
            move_ep.getPostion();
            int x = move_ep.getBlankPos_x(), y = move_ep.getBlankPos_y();
            move_ep.print();
            searched_list.add(move_ep);
            if (move_ep.isEquals(target_ep)) {
                System.out.println("有界深度优先搜索成功！");
                return;
            }
            if (depth < maxDepth - 1) {
                depth++;
                EightPuzzle temp = null;
                temp = move_ep.depthClone();
                if (EightPuzzleOperator.canMove(x, y, 1)) {
                    temp = EightPuzzleOperator.movePosition(move_ep, 1);
                    temp.setDepth(depth);
                    if (!searched_list.contains(temp) ||
(searched_list.contains(temp)
                            &&
searched_list.get(searched_list.indexOf(temp)).getDepth() != temp.getDepth())) {
                        ep_stack.push(temp);
                    }
                }
                if (EightPuzzleOperator.canMove(x, y, 2)) {
                    temp = EightPuzzleOperator.movePosition(move_ep, 2);
                    temp.setDepth(depth);
                    if (!searched_list.contains(temp) ||
(searched_list.contains(temp)
                            &&
searched_list.get(searched_list.indexOf(temp)).getDepth() != temp.getDepth())) {
                        ep_stack.push(temp);
                    }
                }
                if (EightPuzzleOperator.canMove(x, y, -1)) {
                    temp = EightPuzzleOperator.movePosition(move_ep, -1);
                    temp.setDepth(depth);
                    if (!searched_list.contains(temp) ||
```

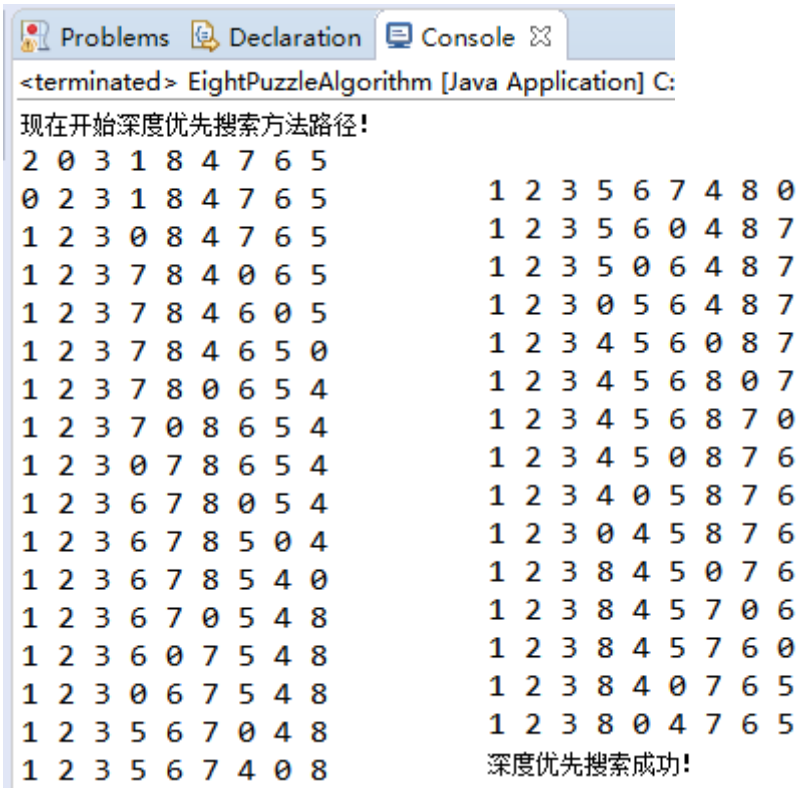
```
(searched_list.contains(temp)
                        &&
searched_list.get(searched_list.indexOf(temp)).getDepth() != temp.getDepth())) {
                    ep_stack.push(temp);
                }
            }
            if (EightPuzzleOperator.canMove(x, y, -2)) {
                temp = EightPuzzleOperator.movePosition(move_ep, -2);
                temp.setDepth(depth);
                if (!searched_list.contains(temp) ||
(searched_list.contains(temp)
                        &&
searched_list.get(searched_list.indexOf(temp)).getDepth() != temp.getDepth())) {
                    ep_stack.push(temp);
                }
            }
        }
    }
    System.out.println("有界深度优先搜索失败！");
}

// 宽度（广度）优先搜索实现
public void breadthFirstSearch() {
    if (!searched_list.isEmpty())
        searched_list.clear();
    System.out.println("\n现在开始广度优先搜索方法路径！");
    while (!ep_queue.isEmpty()) {
        EightPuzzle move_ep = ep_queue.poll();
        depth = move_ep.getDepth();
        move_ep.getPostion();
        int x = move_ep.getBlankPos_x(), y = move_ep.getBlankPos_y();
        move_ep.print();
        searched_list.add(move_ep);
        if (move_ep.isEquals(target_ep)) {
            System.out.println("广度优先搜索成功！");
            return;
        }
        depth++;
        EightPuzzle temp = null;
        temp = move_ep.depthClone();

        if (EightPuzzleOperator.canMove(x, y, 1)) {
```

```
                temp = EightPuzzleOperator.movePosition(move_ep, 1);
                temp.setDepth(depth);
                if (!searched_list.contains(temp)) {
                    ep_queue.offer(temp);
                }
            }
            if (EightPuzzleOperator.canMove(x, y, 2)) {
                temp = EightPuzzleOperator.movePosition(move_ep, 2);
                temp.setDepth(depth);
                if (!searched_list.contains(temp)) {
                    ep_queue.offer(temp);
                }
            }
            if (EightPuzzleOperator.canMove(x, y, -1)) {
                temp = EightPuzzleOperator.movePosition(move_ep, -1);
                temp.setDepth(depth);
                if (!searched_list.contains(temp)) {
                    ep_queue.offer(temp);
                }
            }
            if (EightPuzzleOperator.canMove(x, y, -2)) {
                temp = EightPuzzleOperator.movePosition(move_ep, -2);
                temp.setDepth(depth);
                if (!searched_list.contains(temp)) {
                    ep_queue.offer(temp);
                }
            }
        }
        System.out.println("广度优先搜索失败！");
    }
```

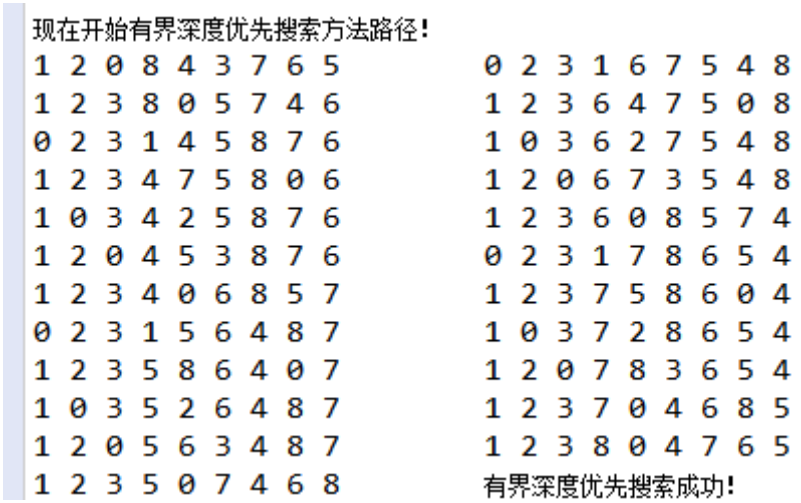
六、测试结果及说明

输入：

深度优先算法结果：



有界深度优先算法结果（界限为 4）：



广度优先算法结果：

```
现在开始广度优先搜索方法路径!
2 0 3 1 8 4 7 6 5
2 3 0 1 8 4 7 6 5
2 8 3 1 0 4 7 6 5
0 2 3 1 8 4 7 6 5
2 3 4 1 8 0 7 6 5
2 8 3 1 4 0 7 6 5
2 8 3 1 6 4 7 0 5
2 8 3 0 1 4 7 6 5
1 2 3 0 8 4 7 6 5
2 3 4 1 8 5 7 6 0
2 3 4 1 0 8 7 6 5
2 8 0 1 4 3 7 6 5
2 8 3 1 4 5 7 6 0
2 8 3 1 6 4 7 5 0
2 8 3 1 6 4 0 7 5
0 8 3 2 1 4 7 6 5
2 8 3 7 1 4 0 6 5
1 2 3 8 0 4 7 6 5
广度优先搜索成功!
```

七、实验体会

通过对状态空间搜索的练习，我对这种搜索算法思想理解的更加深入了，尤其是对深度优先、广度优先的思想理解的更加深入。这个八数码问题只是生活中千千万万的问题中的一个代表，我们还有很多问题都可以用状态空间搜索的方法去解决，当数据量小的时候，我们可以自己模拟变换，但是数据量很大的时候，靠人来操作就很困难了，所以利用计算机是必要的，计算机的计算能力很强，我们要好好利用，至于如何利用才是我们应该去深究的。

附、源码仓库网址

https://github.com/GuoWenhao1996/Artificial-Intelligence

重庆交通大学计算机与信息学院

验 证 性 实 验 报 告

班　　级：计科　专业 14 级 1 班

学　　号：631406010109

姓　　名：郭文浩

实验项目名称：启发式搜索

实验项目性质：验证性实验

实验所属课程：人工智能

实验室(中心)：软件中心实验室（语音楼 8 楼）

指 导 教 师 ：朱振国

实验完成时间：2017 年 6 月 1 日

评阅意见：
实验成绩：　　签名：　　年　月　日

一、实验目的

理解和掌握 A*算法。

二、实验内容及要求

㈠ 实验内容：在 8 数码问题中，利用策略函数判断搜索，并使用 A*算法减少搜索目标。

㈡ 实验要求：用选定的编程语言编写程序，利用不同的搜索策略进行状态空间搜索（如宽度优先搜索、深度优先搜索、有界深度优先搜索等）。

三、实验设备及软件

㈠ 实验设备：重庆交通大学语音楼八楼机房的计算机和自己的笔记本。

㈡ 实验软件：eclipse。

四、设计方案

㈠ 题目

利用启发式搜索算法解决八数码问题

㈡ 设计的主要思路

1、算法思想

估价函数是搜索特性的一种数学表示，是指从问题树根节点到达目标节点所要耗费的全部代价的一种估算，记为 f(n)。估价函数通常由两部分组成，其数学表达式为 f(n)=g(n)+h(n)。其中 f(n) 是节点 n 从初始点到目标点的估价函数，g(n) 是在状态空间中从初始节点到 n 节点的实际代价，h(n)是从 n 到目标节点最佳路径的估计代价。保证找到最短路径（最优解）的条件，关键在于估价函数 h(n)的选取。估价值 h(n)<= n 到目标节点的距离实际值，这种情况下，搜索的点数多，搜索范围大，效率低。但能得到最优解。如果估价值>实际值，搜索的点数少，搜索范围小，效率高，但不能保证得到最优解。

搜索中利用启发式信息，对当前未扩展结点根据设定的估价函数值选取离目标最近的结点进行扩展，从而缩小搜索空间，更快的得到最优解，提高效率。

2、启发函数

进一步考虑当前结点与目标结点的距离信息，令启发函数 h (n)为当前 8 个数字位与目标结点对应数字位距离和（不考虑中间路径），且对于目标状态有 h (t) = 0，对于结点 m 和 n （n 是 m 的子结点） 有 h (m) － h (n) <= 1 = Cost (m, n) 满足单调限制条件。

3、算法介绍

A*（A-Star)算法是一种静态路网中求解最短路最有效的方法。A star 算法在静态路网中的应用。对于几何路网来说，可以取两节点间欧几理德距离（直线距离）做为估价值，即 f=g(n)+sqrt((dx-nx)*(dx-nx)+(dy-ny)*(dy-ny))；这样估价函数 f 在 g 值一定的情况下，会或多或少的受估价值 h 的制约，节点距目标点近，h 值小，f 值相对就小，能保证最短路的搜索向终点的方向进行。明显优于盲目搜索策略。

㈢ 主要功能

将初始状态与目标状态写定在程序里，然后进行模拟启发式搜索，显示搜索过程中的变换，并统计步数和时间。

五、主要代码

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;

@SuppressWarnings("rawtypes")
public class EightPuzzle_AStart implements Comparable {
    private int[] num = new int[9];
    private int depth; // 当前的深度即走到当前状态的步骤
    private int evaluation; // 从起始状态到目标的最小估计值
    private int misposition; // 到目标的最小估计
    private EightPuzzle_AStart parent; // 当前状态的父状态
    public int[] getNum() {
        return num;
    }
    public void setNum(int[] num) {
        this.num = num;
    }
    public int getDepth() {
        return depth;
```

```
}
public void setDepth(int depth) {
    this.depth = depth;
}
public int getEvaluation() {
    return evaluation;
}
public void setEvaluation(int evaluation) {
    this.evaluation = evaluation;
}
public int getMisposition() {
    return misposition;
}
public void setMisposition(int misposition) {
    this.misposition = misposition;
}
public EightPuzzle_AStart getParent() {
    return parent;
}
public void setParent(EightPuzzle_AStart parent) {
    this.parent = parent;
}
public boolean isTarget(EightPuzzle_AStart target) {
    return Arrays.equals(getNum(), target.getNum());
}
public void init(EightPuzzle_AStart target) {
    int temp = 0;
    for (int i = 0; i < 9; i++) {
        if (num[i] != target.getNum()[i])
            temp++;
    }
    this.setMisposition(temp);
    if (this.getParent() == null) {
        this.setDepth(0);
    } else {
        this.depth = this.parent.getDepth() + 1;
    }
    this.setEvaluation(this.getDepth() + this.getMisposition());
}
public boolean isSolvable(EightPuzzle_AStart target) {
    int reverse = 0;
    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < i; j++) {
            if (num[j] > num[i])
```

```
                    reverse++;
                if (target.getNum()[j] > target.getNum()[i])
                    reverse++;
            }
        }
        if (reverse % 2 == 0)
            return true;
        return false;
    }
    @Override
    public int compareTo(Object o) {
        EightPuzzle_AStart c = (EightPuzzle_AStart) o;
        return this.evaluation - c.getEvaluation();// 默认排序为f(n)由小到
大排序
    }
    public int getZeroPosition() {
        int position = -1;
        for (int i = 0; i < 9; i++) {
            if (this.num[i] == 0) {
                position = i;
            }
        }
        return position;
    }
    public int isContains(ArrayList<EightPuzzle_AStart> open) {
        for (int i = 0; i < open.size(); i++) {
            if (Arrays.equals(open.get(i).getNum(), getNum())) {
                return i;
            }
        }
        return -1;
    }
    public boolean isMoveUp() {
        int position = getZeroPosition();
        if (position <= 2) {
            return false;
        }
        return true;
    }
    public boolean isMoveDown() {
        int position = getZeroPosition();
        if (position >= 6) {
            return false;
        }
```

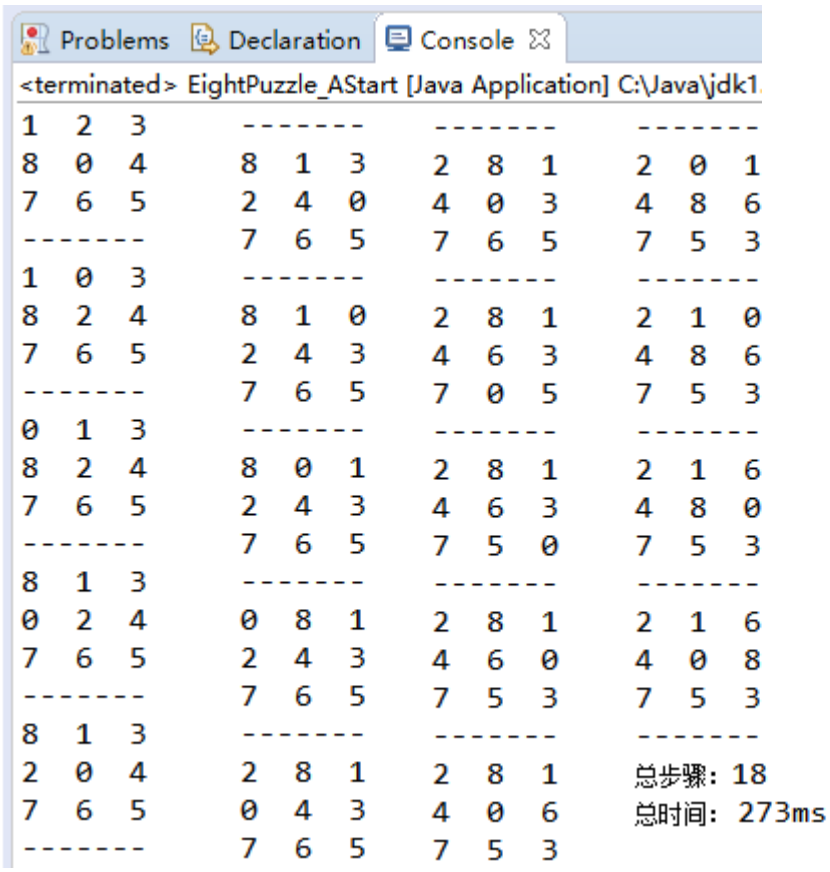
```
        return true;
    }
    public boolean isMoveLeft() {
        int position = getZeroPosition();
        if (position % 3 == 0) {
            return false;
        }
        return true;
    }
    public boolean isMoveRight() {
        int position = getZeroPosition();
        if ((position) % 3 == 2) {
            return false;
        }
        return true;
    }
    public EightPuzzle_AStart moveUp(int move) {
        EightPuzzle_AStart temp = new EightPuzzle_AStart();
        int[] tempnum = (int[]) num.clone();
        temp.setNum(tempnum);
        int position = getZeroPosition(); // 0的位置
        int p = 0; // 与0换位置的位置
        switch (move) {
        case 0:
            p = position - 3;
            temp.getNum()[position] = num[p];
            break;
        case 1:
            p = position + 3;
            temp.getNum()[position] = num[p];
            break;
        case 2:
            p = position - 1;
            temp.getNum()[position] = num[p];
            break;
        case 3:
            p = position + 1;
            temp.getNum()[position] = num[p];
            break;
        }
        temp.getNum()[p] = 0;
        return temp;
    }
```

```
    public void print() {
        for (int i = 0; i < 9; i++) {
            if (i % 3 == 2) {
                System.out.println(this.num[i]);
            } else {
                System.out.print(this.num[i] + "  ");
            }
        }
    }
    public void printRoute() {
        EightPuzzle_AStart temp = null;
        int count = 0;
        temp = this;
        while (temp != null) {
            temp.print();
            System.out.println("-------");
            temp = temp.getParent();
            count++;
        }
        System.out.println("总步骤: " + (count - 1));
    }
    public void operation(ArrayList<EightPuzzle_AStart> open,
ArrayList<EightPuzzle_AStart> close, EightPuzzle_AStart parent,
            EightPuzzle_AStart target) {
        if (this.isContains(close) == -1) {
            int position = this.isContains(open);
            if (position == -1) {
                this.parent = parent;
                this.init(target);
                open.add(this);
            } else {
                if (this.getDepth() < open.get(position).getDepth()) {
                    open.remove(position);
                    this.parent = parent;
                    this.init(target);
                    open.add(this);
                }
            }
        }
    }
    @SuppressWarnings("unchecked")
    public static void main(String args[]) {
        // 定义open表
        ArrayList<EightPuzzle_AStart> open = new
```

```
ArrayList<EightPuzzle_AStart>();
        ArrayList<EightPuzzle_AStart> close = new
ArrayList<EightPuzzle_AStart>();
        EightPuzzle_AStart start = new EightPuzzle_AStart();
        EightPuzzle_AStart target = new EightPuzzle_AStart();

        int stnum[] = { 2, 1, 6, 4, 0, 8, 7, 5, 3 };
        int tanum[] = { 1, 2, 3, 8, 0, 4, 7, 6, 5 };
        start.setNum(stnum);
        target.setNum(tanum);
        long startTime = System.currentTimeMillis(); // 获取开始时间
        if (start.isSolvable(target)) {
            // 初始化初始状态
            start.init(target);
            open.add(start);
            while (open.isEmpty() == false) {
                Collections.sort(open); // 按照evaluation的值排序
                EightPuzzle_AStart best = open.get(0); // 从open表中取出最
小估值的状态并移除open表
                open.remove(0);
                close.add(best);
                if (best.isTarget(target)) {
                    // 输出
                    best.printRoute();
                    long end = System.currentTimeMillis(); // 获取结束时间
                    System.out.println("总时间:  " + (end - startTime) +
"ms");

                    System.exit(0);
                }
                int move;
                // 由best状态进行扩展并加入到open表中
                // 0的位置上移之后状态不在close和open中设定best为其父状态，并
初始化f(n)估值函数
                if (best.isMoveUp()) {
                    move = 0;
                    EightPuzzle_AStart up = best.moveUp(move);
                    up.operation(open, close, best, target);
                }
                // 0的位置下移之后状态不在close和open中设定best为其父状态，并
初始化f(n)估值函数
                if (best.isMoveDown()) {
                    move = 1;
                    EightPuzzle_AStart up = best.moveUp(move);
                    up.operation(open, close, best, target);
```

```
                }
                // 0的位置左移之后状态不在close和open中设定best为其父状态，并初始化f(n)估值函数
                if (best.isMoveLeft()) {
                    move = 2;
                    EightPuzzle_AStart up = best.moveUp(move);
                    up.operation(open, close, best, target);
                }
                // 0的位置右移之后状态不在close和open中设定best为其父状态，并初始化f(n)估值函数
                if (best.isMoveRight()) {
                    move = 3;
                    EightPuzzle_AStart up = best.moveUp(move);
                    up.operation(open, close, best, target);
                }
            }
        } else
            System.out.println("没有解，请重新输入。");
    }
}
```

六、测试结果及说明



七、实验体会

这次对启发式搜索做了实验，最大的感受就是启发式搜索的效率真的很高，排除掉自己代码的一些客观因素，启发式搜索的效率比状态空间搜索的效率高太多了，它可以很快的找到最优的解，是一个特别好的算法。但是随着高效率的优点，带来的是高不稳定性，它对于一些明明是有解的情况却解不出来，可能是我代码实现的有问题。不过我个人觉得一个事物它带来了优点那必定会引入缺陷，这是不可改变的。

另外，人工智能的实验课程也到此结束了。人工智能这门课程给我的算法带来了很大的提高，不论是从思想方面还是编码能力方面都得到了很大的进步。学无止境，加油。

附、源码仓库网址

https://github.com/GuoWenhao1996/Artificial-Intelligence