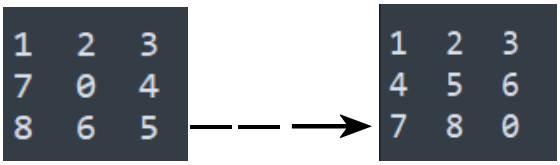
实验一　A*算法解决八数码问题

学号：2017526019　　班级：2015211307　　姓名：刘禾子

一、 问题描述

八数码问题就是在一个 3*3 的矩阵中摆放 1-8 一共八个数字，还有一个空余的位置（数字 0）用于移动来改变当前的位置来达到最终的状态，如下图所示：



二、 算法思想

A*算法是一种静态路网中求解最短路径最有效的直接搜索方法也是一种启发性的算法，也是解决许多搜索问题的有效算法。算法中的距离估算值与实际值越接近，最终搜索速度越快。启发中的估价是用估价函数表示的，如 f（n）=g(n)+h(n)，其中：

f(n):是节点 n 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；

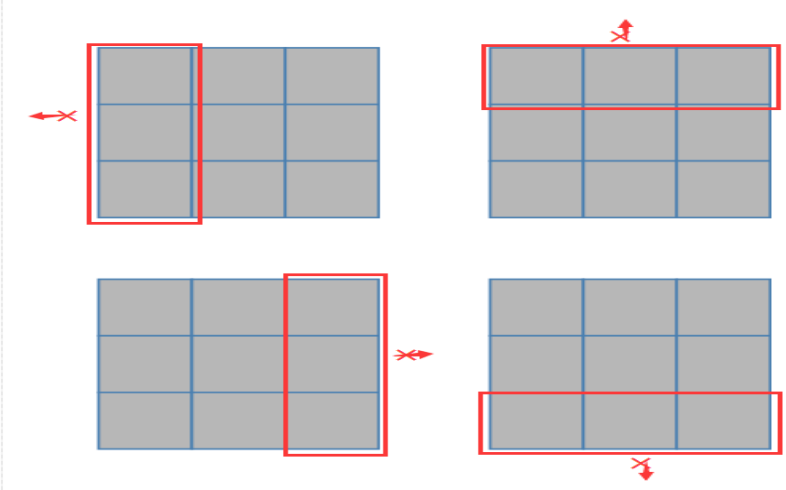
h(n):是从 n 到目标节点最佳路径的估计代价，其中最重要的是 h(n)函数，要求 h(n)<h’(n)，其中 h’(n)为实际中当前状态要到达目标状态的步骤数，在这里表示与目标八数码状态所不同的方格数。

三、 设计过程

首先，将八数码问题降维，移动数字就是相当于移动空格。这样问题就简化为空格的移动，空格的状态只有 4 种：上、下、左、右。然而在八数码问题中并不是每次空格的移动都有四种状态，需要判断在当前位置信息是否可以移动才能移动，还需去掉当前状态的父状态（若当前移动到父状态则相当于回退了一步）。

其次，需要关注的是给定的初始化状态是否能够通过移动而到达目标状态，这涉及到数学的定则：若初始状态和目标状态的逆序值同为奇数或同为偶数则可以通过有限次数的移动到达目标状态，否则无解。

最后，根据下图明确空格的几种可能的移动方式：



四、 实验结果

```
"C:\Program Files\Java\jdk1.8.0_112\bin\java"
1  2  3
7  0  4
8  6  5
----------分割线----------
1  2  3
7  4  0
8  6  5
----------分割线----------
1  2  3
7  4  5
8  6  0
----------分割线----------
1  2  3
7  4  5
8  0  6
----------分割线----------
1  2  3
7  4  5
0  8  6
----------分割线----------
1  2  3
0  4  5
7  8  6
----------分割线----------
1  2  3
4  0  5
7  8  6
----------分割线----------
1  2  3
4  5  0
7  8  6
----------分割线----------
1  2  3
4  5  6
7  8  0
----------分割线----------
步骤数：8
```

五、 源代码

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

public class EightPuzzle 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 parent;  //当前状态的父状态
    private int[] getNum() {
        return num;
    }
    private void setNum(int[] num) {
        this.num = num;
    }
    private int getDepth() {
        return depth;
    }
    private void setDepth(int depth) {
        this.depth = depth;
    }
    private int getEvaluation() {
        return evaluation;
    }
    private void setEvaluation(int evaluation) {
        this.evaluation = evaluation;
    }
    private int getMisposition() {
        return misposition;
    }
    private void setMisposition(int misposition) {
        this.misposition = misposition;
    }
    private EightPuzzle getParent() {
        return parent;
    }
    public void setParent(EightPuzzle parent) {
        this.parent = parent;
    }

    /**
     * 判断当前状态是否为目标状态
     * @param target
     * @return
     */
    private boolean isTarget(EightPuzzle target){
        return Arrays.equals(getNum(), target.getNum());
    }
```

```
    /**
     * 求 f(n) = g(n)+h(n);
     * 初始化状态信息
     * @param target
     */
    private void init(EightPuzzle 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());
    }

    /**
     * 求逆序值并判断是否有解
     * @param target
     * @return 有解: true 无解: false
     */
    private boolean isSolvable(EightPuzzle 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++;
            }
        }
        return reverse % 2 == 0;
    }
    @Override
    public int compareTo(Object o) {
        EightPuzzle c = (EightPuzzle) o;
        return this.evaluation-c.getEvaluation();//默认排序为 f(n)由小到
大排序
    }
    /**
     * @return 返回 0 在八数码中的位置
     */
    private int getZeroPosition(){
        int position = -1;
        for(int i=0;i<9;i++){
            if(this.num[i] == 0){
                position = i;
```

```
        }
    }
    return position;
}
/**
 *
 * @param open    状态集合
 * @return 判断当前状态是否存在于open表中
 */
private int isContains(ArrayList<EightPuzzle> open){
    for(int i=0;i<open.size();i++){
        if(Arrays.equals(open.get(i).getNum(), getNum())){
            return i;
        }
    }
    return -1;
}
/**
 *
 * @return 小于3的不能上移返回false
 */
private boolean isMoveUp() {
    int position = getZeroPosition();
    return position > 2;
}
/**
 *
 * @return 大于6返回false
 */
private boolean isMoveDown() {
    int position = getZeroPosition();
    return position < 6;
}
/**
 *
 * @return 0, 3, 6返回false
 */
private boolean isMoveLeft() {
    int position = getZeroPosition();
    return position % 3 != 0;
}
/**
 *
 * @return 2, 5, 8不能右移返回false
 */
private boolean isMoveRight() {
    int position = getZeroPosition();
    return (position) % 3 != 2;
}
/**
 *
```

```
     * @param move 0: 上, 1: 下, 2: 左, 3: 右
     * @return 返回移动后的状态
     */
    private EightPuzzle moveUp(int move){
        EightPuzzle temp = new EightPuzzle();
        int[] tempnum = 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;
    }
    /**
     * 按照八数码的格式输出
     */
    private 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]+"  ");
            }
        }
    }
    /**
     * 反序列的输出状态
     */
    private void printRoute(){
        EightPuzzle temp;
        int count = 0;
        temp = this;
        while(temp!=null){
            temp.print();
```

```
            System.out.println("----------分割线----------");
            temp = temp.getParent();
            count++;
        }
        System.out.println("步骤数: "+(count-1));
    }
    /**
     *
     * @param open open表
     * @param close close表
     * @param parent 父状态
     * @param target 目标状态
     */
    private void operation(ArrayList<EightPuzzle> open,
ArrayList<EightPuzzle> close, EightPuzzle parent, EightPuzzle 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);
                }
            }
        }
    }
 /*主函数*/
    public static void main(String args[]){
        //定义open表
        ArrayList<EightPuzzle> open = new ArrayList<>();
        ArrayList<EightPuzzle> close = new ArrayList<>();
        EightPuzzle start = new EightPuzzle();
        EightPuzzle target = new EightPuzzle();
        int stnum[] = {1,2,3,4,5,6,7,8,0};
        int tanum[] = {1,2,3,7,0,4,8,6,5};

        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()){
                Collections.sort(open);            //按照evaluation的值
排序
```

```
                EightPuzzle 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("程序运行时间：  "+(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 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 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 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 up = best.moveUp(move);
                    up.operation(open, close, best, target);
                }

            }
        }else
            System.out.println("没有解，请重新输入。");
    }
 }
```