# 八数码问题

## 一：八数码问题：

八数码难题也称九宫问题，它是在3×3的方格棋盘上，分别放置了表有数字1、2、3、4、5、6、7、8的八张牌，初始状态S0，目标状态Sg，要求程序能输入任意的初始状态和目标状态，要求通过空格来移动八张牌使得棋盘由初始状态到达目标状态。移动规则为：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。

在开始求解之前，我们先判断该八数码问题是否有解，判断方法如下：

八数码问题的一个状态实际上是0~9的一个排列，空格用0表示，对于任意给定的初始状态和目标状态，不一定有解，也就是说从初始状态不一定能到达目标状态。因为排列有奇排列和偶排列两类，排列只能在同类排列之间转化，而从奇排列不能转化成偶排列或相反。

如果一个数字0~8的随机排列871526340，用F(X)表示数字X前面比它小的数的个数，全部数字的F(X)之和为Y=∑(F(X))，如果Y为奇数则称原数字的排列是奇排列，如果Y为偶数则称原数字的排列是偶排列。

例：871526340这个排列的Y=0+0+0+1+1+3+2+3+0=10，10是偶数，所以是偶排列。

871625340，Y=0+0+0+1+1+2+2+3+0=9，9是奇数，所以是奇排列。

因此，可以在运行程序前检查初始状态和目标状态的排列是否相同，相同则问题可解，接着采用搜索算法求解，否则无解。

## 二：试验代码：

**package** eight\_figure\_puzzle;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.Comparator;

**import** java.util.HashMap;

**import** java.util.List;

**import** java.util.PriorityQueue;

**import** java.util.Queue;

**import** java.util.Scanner;

**public** **class** eight\_figure\_puzzle {

**static** **int** *N* = 0;

**static** **int**[][] *MT* = **new** **int**[3][3];

**static** **int**[][] *endMT* = **new** **int**[3][3];

**static** HashMap<Integer, **int**[]> *map* = **new** HashMap<>();

**static** **int**[][] *dir* = {{0, 1}, {0, -1}, {-1, 0}, {1, 0}};

**static** List<**int**[][]> *marke* = **new** ArrayList<**int**[][]>();

**static** **void** initMap(){

**int**[][] point = {{1,1},{0,0},{0,1},{0,2},{1,2},{2,2},{2,1},{2,0},{1,0}};

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

*map*.put(i, point[i]);

}

}

**static** **public** **class** node **implements** Cloneable{

**int** x;

**int** y;

**int** g;

**int** h;

**int** step;

**int**[][] mt = **new** **int**[*N*][];

List<**int**[][]> path = **new** ArrayList<**int**[][]>();

**public** node(**int** x, **int** y, **int** g, **int** h, **int** step, **int**[][] mt, List<**int**[][]> path) {

**super**();

**this**.x = x;

**this**.y = y;

**this**.g = g;

**this**.h = h;

**this**.step = step;

**this**.mt = mt;

**this**.path = path;

}

@Override

**public** String toString() {

**return** "node [x=" + x + ", y=" + y + ", g=" + g + ", h=" + h + ", step=" + step + ", mt="

+ Arrays.*toString*(mt) + "]";

}

**public** Object clone(){

node nd = **null**;

**try** {

nd = (node) **super**.clone();

} **catch** (CloneNotSupportedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

nd.mt = **new** **int**[3][];

**for**(**int** r=0; r<*N*; r++){

nd.mt[r] = **this**.mt[r].clone();

}

nd.path = **new** ArrayList<**int**[][]>();

nd.path.addAll(**this**.path);

**return** nd;

}

}

**static** Comparator<node> *cmp* = **new** Comparator<node>() {

@Override

**public** **int** compare(node o1, node o2) {

// **TODO** Auto-generated method stub

**return** (o1.g+o1.h) - (o2.g+o2.h);

}

};

**static** **boolean** input\_date(){

@SuppressWarnings("resource")

Scanner in = **new** Scanner(System.***in***);

// System.out.println("请输入八数码矩阵大小：");

// N = in.nextInt();

*N* = 3;

System.***out***.println("请输入初始状态(0代表空白位置)：");

**for**(**int** i=0; i<*N*; i++){

*MT*[i][0] = in.nextInt();

*MT*[i][1] = in.nextInt();

*MT*[i][2] = in.nextInt();

}

System.***out***.println("请输入目标状态(0代表空白位置)：");

**for**(**int** i=0; i<*N*; i++){

*endMT*[i][0] = in.nextInt();

*endMT*[i][1] = in.nextInt();

*endMT*[i][2] = in.nextInt();

}

**for**(**int** i=0; i<*N*; i++){

**for**(**int** j=0; j<*N*; j++){

**int**[] temp = {i,j};

*map*.put(*endMT*[i][j], temp);

}

}

**int** st = 0;

**int** et = 0;

**int**[] startNum = **new** **int**[*N*\**N*];

**int**[] endNum = **new** **int**[*N*\**N*];

**int** cnt1 = 0;

**int** cnt2 = 0;

**for**(**int** i=0; i<*N*; i++){

**for**(**int** j=0; j<*N*; j++){

**if**(*MT*[i][j]!=0){

startNum[cnt1++] = *MT*[i][j];

}

**if**(*endMT*[i][j]!=0){

endNum[cnt2++] = *endMT*[i][j];

}

}

}

**for**(**int** i=*N*\**N*-2; i>=0; i--){

**for**(**int** j=i-1; j>=0; j--){

**if**(startNum[i]>startNum[j]){

st++;

}

**if**(endNum[i]>endNum[j]){

et++;

}

}

}

System.***out***.println(Integer.*toString*(st)+" "+Integer.*toString*(et));

**if**(st%2==et%2) **return** **true**;

**return** **false**;

}

**static** **int** A\_star(**int**[][] MT){

**int** x0=0,y0 = 0;

**for**(x0=0; x0<*N*; x0++){

**boolean** flag = **false**;

**for**(y0=0; y0<*N*; y0++){

**if**(MT[x0][y0]==0){

flag = **true**;

**break**;

}

}

**if**(flag) **break**;

}

Queue<node> q = **new** PriorityQueue<node>(*cmp*);

**int**[][] curmt = **new** **int**[*N*][];

**int**[][] markemt = **new** **int**[*N*][];

**for**(**int** r=0; r<*N*; r++){

curmt[r] = MT[r].clone();

}

**for**(**int** r=0; r<*N*; r++){

markemt[r] = MT[r].clone();

}

List<**int**[][]> path = **new** ArrayList<**int**[][]>();

path.add(MT);

node cur = **new** node(x0, y0, 0, 0, 0, curmt, path);

*marke*.add(markemt);

q.add(cur);

**while**(!q.isEmpty()){

cur = (node) q.poll().clone();

**boolean** tag = **false**;

**for**(**int** i=0; i<*N*; i++){

**for**(**int** j=0; j<*N*; j++){

**if**(cur.mt[i][j]!=*endMT*[i][j]){

tag = **true**;

}

}

}

**if**(!tag){

**for**(**int** v=0; v<cur.path.size(); v++){

System.***out***.println("---------------第"+Integer.*toString*(v)+"步---------------");

**for**(**int** i=0; i<*N*; i++){

**for**(**int** j=0; j<*N*; j++){

System.***out***.print(Integer.*toString*(cur.path.get(v)[i][j])+" ");

}

System.***out***.println();

}

System.***out***.println();

}

**return** cur.step;

}

**for**(**int** i=0; i<4; i++){

node next = (node) cur.clone();

next.x = cur.x + *dir*[i][0];

next.y = cur.y + *dir*[i][1];

**if**(next.x>=0 && next.x<*N* && next.y>=0 && next.y<*N*){

next.mt[cur.x][cur.y] = next.mt[next.x][next.y];

next.mt[next.x][next.y] = 0;

**boolean** mark = **false**;

**for**(**int** c=0; c<*marke*.size(); c++){

**int** x=0,y=0;

**for**(x=0; x<*N*; x++){

**for**(y=0; y<*N*; y++){

**if**(*marke*.get(c)[x][y]!=next.mt[x][y]){

**break**;

}

}

**if**(y<*N*) **break**;

}

**if**(x==*N* && y==*N*) mark = **true**;

}

**if**(!mark){

next.step++;

next.g++;

next.path.add(next.mt);

**int** count = 0;

**for**(**int** row=0; row<*N*; row++){

**for**(**int** cow=0; cow<*N*; cow++){

**if**(cow!=0 && next.mt[row][cow]!=*endMT*[row][cow]){

count += Math.*abs*(row-*map*.get(next.mt[row][cow])[0]) + Math.*abs*(cow-*map*.get(next.mt[row][cow])[1]);

}

}

}

next.h = count;

**int**[][] newmt = **new** **int**[*N*][];

**for**(**int** r=0; r<*N*; r++){

newmt[r] = next.mt[r].clone();

}

*marke*.add(newmt);

q.add((node) next.clone());

}

}

}

}

**return** 0;

}

**public** **static** **void** main(String[] args) {

System.***out***.println("-------------------------八数码A\*算法实现------------------------");

*initMap*();

**boolean** flag = *input\_date*();

**if**(!flag){

System.***out***.println("问题无解！");

}

**else**{

**int** ans = *A\_star*(*MT*);

System.***out***.println("移动步数："+Integer.*toString*(ans));

}

}

}

## 三．试验结果：





