实验报告

——A*算法求解八数码问题

姓名：鲁沛　　学号：18342066　　日期：2021.1.10
组号：19
组长：车春江
个人分工：A*算法的全部内容

摘要：解决的问题是八数码问题，使用的算法是 A*算法，得到的结论是 A*算法求解八数码问题效率较高，其估价函数很关键，明确影响着其求解效率。

一、 导言

1. 八数码问题：

八数码问题也称为九宫格问题。在 3×3 的棋盘，摆有 8 个棋子，每个棋子上标有 1 至 8 的某一数字，不同棋子上标的数字不相同。棋盘上还有一个空格，与空格相邻的棋子可以移到空格中。要求解决的问题是: 给出一个初始状态和一个目标状态，找出一种从初始转变成目标状态的移动棋子步数最少的移动步骤。

2. A*算法：

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，也是许多其他问题的常用启发式算法。其之所以是一种启发式算法是因为其需要使用一个估价函数:

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

保证找到最短路径（最优解的）条件，关键就在于估价函数 f(n)的选取（或者说 h(n)的选取）。

二、 实验过程

1. A 算法与 A*算法

在或图通用搜索算法中，将启发式函数的形式定义为 f(n) = g(n) + h(n)。
A 算法的启发式函数中，g(n)表示从S_0到 n 点的搜索费用的估计，因为 n 为当前节点，搜索已到达 n 点，所以 g(n)可以计算出。h(n)表示从 n 到S_g接近程度的估计，因为尚未找到解路径，所以 h(n)仅仅是估计值。
若进一步规定 h(n) >= 0，并且定义:

$$f^*(n) = g^*(n) + h^*(n)$$

式中，$f^*(n)$表示S_0经点 n 到S_g最优路径的搜索费用，也有人将$f^*(n)$定义为实际最小搜索费用；$g^*(n)$为S_0到 n 的实际最小费用；$h^*(n)$为 n 到S_g的实际最小费用的估计。

当要求 h(n) <= $h^*(n)$，就称这种 A 算法为 A*算法。

2. A*算法的流程

设S_0为初始状态，S_g为目标状态：

1) open = {S_0};
2) closed = {};
3) 如果 open = {}，失败退出;
4) 在 open 表上取出 f(n)值最小的节点 n，n 放到 closed 表中，即：
   f(n) = g(n) + h(n)，h < h^*;
5) 若 n∈S_g，则成功退出;
6) 产生 n 的一切后继，将后继中不是 n 的前驱节点的一切点构成集合 M;
7) 对 M 中的元素 P，分别做两类处理：
   a) 若 P 不属于 G，则对 P 进行估计并加入 open 表，计入 G 和 Tree;
   b) 若 P∈G，则决定是否更改 Tree 中 P 到 n 的指针，并且更改 P 的子节点 n 的指针和费用;

转第(3)步。

3. 实现算法的程序主要流程

使用 java 按照面向对象的思想编写。

设计拼图节点类 JigsawNode：

成员变量如下：

```
private int []nodeState                    //拼图节点本身
private int depth                          //从初始状态到当前状态的步数
private JigsawNode parent                   //当前状态的上一个状态
private int estimatedValue                 //代价估计值
```

nodeState 使用一维整数数组来存，如下状态被定义为{9，1，2，3，4，5，6，7，8，0}：

1	2	3
4	5	6
7	8	

即每个数组大小为 10，第一个数字代表 0 所在的空格的位置，接下来的 9 个数字分别为 9 个格子中的数字，以此类推。

成员函数如下：

```
public JigsawNode(int[] data)
public JigsawNode(JigsawNode node)
public int[] getNodeState()
```

```
public int getDepth()
public JigsawNode getParent()
public int getEstimatedValue()
public void setEstimatedValue(int value)
public void setParent(JigsawNode parent)
public int hashCode()
public boolean equals(Object obj)
public String toString()                    //将 nodeState 转化为字符串
public boolean move(int direction)          //将空白格向上下左右四个方向移动
public boolean moveUp()
public boolean moveRight()
public boolean moveDown()
public boolean moveLeft()
```

hashcode 和 equal 函数是为了在 Jigsaw 类中使用 set 而重载的，4 个方向的 move 函数都要首先判断能否移动（空白格在边上的情况）。move 函数中 0 代表上移，1 代表右移，2 代表下移，3 代表左移。

设计拼图类 Jigsaw：

成员变量如下：

```
protected JigsawNode begin;                 //开始状态
protected JigsawNode end;                   //目标状态
protected JigsawNode current;               //当前状态
private List<JigsawNode> solution;          //解路径
private int nodeNum;                        //考察过的节点数量
private ArrayList<JigsawNode> open;         //已经生成，但还未生成后继节点的节点
private Set<JigsawNode> closed;             //已经生成且已经考察过的节点
```

solution 中保存的是逆序的解路径，是从终点开始循环找出 parent 得到。

成员函数如下：

```
public Jigsaw()
public Jigsaw(JigsawNode Begin, JigsawNode End)
public JigsawNode getBeginNode()
public JigsawNode getEndNode()
public JigsawNode getCurrentNode()
public int getNodeNum()
public boolean solvability()                //判断是否可解
private int getManhattan(JigsawNode node)   //计算曼哈顿距离
public void estimateValue(JigsawNode Node)  //计算估价函数
public void getSolution()
public final String getPath()               //得到正序的解路径
```

```
public void printResult()                                    //将路径打印在终端
public void printToTxt(FileWriter pw, int num)               //将路径打印入文件中
public boolean Astar()                                       //A*算法本体
```

solvability()实现：

将拼图节点的每个方格中的数字按顺序写为一维数组，求出除 0 外的每个数字之前的比它小的数字的个数之和（逆序和），若拼图初始状态和结束状态的逆序和的奇偶性相同，则此八数码问题可解。

getManhattan()实现：

曼哈顿距离指的是两点在南北方向上的距离加上在东西方向上的距离，即横坐标之差的绝对值加上纵坐标之差的绝对值，此函数计算给定状态和目标状态的每个数字的曼哈顿距离之和。

Astar()实现：

1) 判断该八数码问题是否可解；
2) 设置搜索上限，一旦超过则视为不可解
3) 将初始状态节点加入 open 表和 close 表中；
4) 当 open 表不为空且没有达到搜索上限时，将 open 表按照估价值从小到大排序，取出其中最小的状态节点，将其加入 closed 表，若其等于目标状态，则结束，否则求出其向上下左右 4 个方向移动后生成的节点，对它们：
   a) 若在 closed 表中，忽略；
   b) 若已经在 open 表中，则找到旧节点，比较新旧节点的估价值来决定是否更新路径，即更改此节点的 parent 和估价值；
   c) 若不在 open 表中，则将其加入 open 表。

设计运行类 runner：

此类中包含 main 函数，用于根据给定初始状态和结束状态使用前文设计的 A*算法求解八数码问题，并在终端可视化其求解过程。

设计运行类 runner2：

此类中包含 main 函数，其使用 1000 个随机生成的测例来分别测试 h1(n)和 h2(n)的

效率，并将结果输出在 txt 文档中。

三、 结果分析

1. 实验环境

编程语言：java
编程环境：IntelliJ IDEA Community 2020.3

操作系统：windows 10
处理器：Intel(R) Core(TM) i5-8300H
RAM：8G
GPU：1050Ti

2. 可视化测试（runner 类）

给定开始状态为{5, 1, 2, 3, 4, 0, 6, 7, 5, 8}，结束状态为{9, 1, 2, 3, 4, 5, 6, 7, 8, 0}，输出结果如下：

```
搜索深度为：10
查找解成功，路径如下：
-------------
| 1 | 2 | 3 |
|-----------|
| 4 | 0 | 6 |
|-----------|
| 7 | 5 | 8 |
-------------


-------------
| 1 | 2 | 3 |
|-----------|
| 4 | 5 | 6 |
|-----------|
| 7 | 0 | 8 |
-------------


-------------
| 1 | 2 | 3 |
|-----------|
| 4 | 5 | 6 |
|-----------|
| 7 | 8 | 0 |
-------------
```

3. 两种 h(n)的运行结果（runner2 类）：

h1(n): 放错位置的数字个数
h2(n): 该节点与目标节点之间的曼哈顿距离
以**考察过的节点数量**作为衡量效率的标准
分别用 h1(n)和 h2(n)运行 runner2 10 次，得到如下统计表：

	h1(n)	h2(n)
最好解	1523	1602
最差解	4432	4893
平均值	2496	2613

则 h1(n)的搜索效率略高，h2(n)看似更加复杂专业，实际效果却并没有 h1(n)好。

4. 算法分析

只要此八数码问题有解，则 A*算法一定可以得到最优解，因此解的精度是非常高的。而算法的速度则主要取决于启发式函数的选取，本次实验中选取的两种启发式函数速度一般，其实还有更加复杂的启发式函数可以得到更快的运算速度。这也是本次实验还可以改进的地方：

1) 将启发式函数的 g(n)也换一种形式
2) 多种 h(n)混合在一起计算估价值，h(n)还可以是欧式距离等。

A*算法的优点：A*算法一定能找到最优解；若以搜索的节点数来估计它的效率，则当启发式函数 h 的值单调上升时它的效率只会提高不会降低；满足单调性时，有很理想的搜索路径。

A*算法的缺点：当 h(n)过低估计$h^*(n)$时，有时会显出很高的复杂性。

四、 结论

结论已经在前文中说得很详细了，因此在此谈一谈本次实验中我的心得体会。

八数码问题和 A*算法在上学期的算法课中其实已经有所接触，但一直没有用代码实现。俗话说纸上得来终觉浅，绝知此事要躬行，本次实验写的代码虽然只有几百行，但是让我真正的理解透了什么是 A*算法，并且还锻炼了我 java 编程的能力，收获可以说非常大了。

主要参考文献：

朱福喜 人工智能基础教程（第二版）

张信一，黎燕. 基于 A*算法的八数码问题的程序求解[J]. 现代计算机(专业版), 2003.

史辉，曹闻，朱述龙,等. A*算法的改进及其在路径规划中的应用[J]. 测绘与空间地理信息, 2009.