实验报告

————A*算法

姓名：陈恩婷　　学号：19335015　　日期：2020/9/25

摘要：本实验在 Matlab 中编程实现了用 A*算法求解八数码问题的程序，并通过用不同的输入样例多次运行程序，对比了 h1 与 h2 的搜索效率。

1. 导言

问题描述：编程实现 A*算法求解八数码问题，可视化其求解过程，比较两种启发函数（上课和书上讲到的 h1(n)和 h2(n)）的搜索效率。注意能否达到目标状态的判断方法。

方法：在 Matlab 2020b 中使用 A*算法编写代码，用于求解八数码问题。

2. 实验过程

主程序 EightPuzzle 的主要流程如下：

1. 读入八数码问题
2. 检查八数码问题是否有解：Solvable 函数
3. 用 A*算法求出搜索树；AStar 函数
4. 根据求得的搜索数得到并打印八数码问题的解：printSolution 函数

以下分别介绍上面三个主要函数与两个启发函数 h1 和 h2 的实现思想与方法。

1． Solvable 函数

该函数主要利用以下结论确定八数码问题是否有解：**若两个状态的一维形式的奇偶性相同，则可相互到达，否则不可相互到达。**

关于一维形式：

如原始状态如下：
1 2 3
4 5 6
7 8
则它的逆序为：

1 2 3 4 5 6 7 8 0

逆序和逆序数的定义：在一个排列中，如果一对数的前后位置与大小顺序相反，即前面的数大于后面的数，那么它们就称为一个逆序。 一个排列中逆序的总数就称为这个排列的逆序数。

所以只需要判断初始状态与目标状态的逆序奇偶性是否相同即可。在 Matlab 下计算数组的逆序有一些小技巧，也就是利用 Matlab 本身所支持的 ASCII 字符串比较：

```
>>'8' > '92361'
ans =

     0     1     1     1     1
```

其原理就是通过ASCII编码进行字符级的比较（逐个比较，size不匹配时自动broadcasting）
所以最后可以这样实现：

```
clear all
a = 4132;
b = num2str(a);
total = 0;

for n = 1:length(b)
    total = total + sum(b(n)>b(n:end));% 这里根据改题另一回答最初优化修正
end
```

2. AStar 函数

利用 A*算法求出八数码问题的搜索树。
具体流程：

设S_0：初态，S_g：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最小的结点n，　n放到closed表中；

   f(n)=g(n)+h(n)　　h<=h*

5. 若$n \in S_g$，则成功退出；
6. 产生n的一切后继，将后继中不是n的先辈点的一切点构成集合M
7. 对M中的元素P，分别作两类处理：

   7.1 若P∉G，则P对P进行估计加入open表，记入G和Tree。

   7.2 P∈G，则决定更改Tree中P到n的指针

   并且更改P的子节点n的指针和费用。
8. 转3。

3． printSolution 函数

该函数根据 Astar 函数所求得的搜索树，计算并打印出八数码问题的解，其中树上的每个节点包含三个数据成员：matrix，level 和 parent。主要流程如下：

1. solution = []；
2. 在搜索树中找到目标状态，令 s=该节点，将 s.matrix 放入 solution；
3. 从 s 中读出 s.parent；
4. 在搜索树中找到 matrix 数据成员与 s.parent 相等的节点，令 s=该节点，将 s.matrix 放到 solution 头部；
5. 判断 s.level 是否为 0，是则从头输出 solution，否则重复 2-4 步骤。

4． 启发函数 h1 和 h2

根据课本中的解释，h1 和 h2 的定义如下：

- h_1 = the number of misplaced tiles. For Figure 3.28, all of the eight tiles are out of position, so the start state would have $h_1 = 8$. h_1 is an admissible heuristic because it is clear that any tile that is out of place must be moved at least once.
- h_2 = the sum of the distances of the tiles from their goal positions. Because tiles cannot move along diagonals, the distance we will count is the sum of the horizontal and vertical distances. This is sometimes called the **city block distance** or **Manhattan distance**. h_2 is also admissible because all any move can do is move one tile one step closer to the goal. Tiles 1 to 8 in the start state give a Manhattan distance of

$$h_2 = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18 .$$

实现的方法比较简单，利用 Matlab 中的库函数可以很容易实现它们，这里就不再赘述了。

3. 结果分析

本实验在 Matlab 2020b 上完成。下表是选用 10 种不同的输入求解八数码问题的结果，其中 d 表示解的长度（步骤数）。

	Search Cost (Nodes Generated)	
d	h1	h2
6	7	6
10	37	12
13	149	58
16	432	35
16	529	173
17	866	291
18	1078	179
18	1309	372
20	----	599
24	----	818

如表所示，可以看到搜索代价随解的步骤数增加而增大，启发函数 h2 的代价每次都比 h1 小，而且解越长差距越明显。这验证了课本中 h2 在搜索效率上优于 h1 的结论。

4. 结论

本实验在 Matlab 中编程实现了用 A*算法求解八数码问题的程序，并通过用不同的输入样例多次运行程序，对比了 h1 与 h2 的搜索效率，验证了《人工智能：一种现代的方法》中和课堂上对于其样例差距的结论。

虽然收获颇多，但是本实验的具体实现当然也有一些不足之处。比如由于自己电脑的算力有限，不能像课本中一样对更多不同的输入样例进行尝试再将结果求平均，希望以后有机会在更好的平台上进行测试。

主要参考文献(三五个即可)

1. https://blog.csdn.net/kala0/article/details/52894314
2. https://www.zhihu.com/question/36567748/answer/68049992
3. Artificial Intelligence: A Modern Approach. Third Edition