人工智能-第一次课程作业报告

授课教师：杨旭 作者：徐子航-61520711

1 问题描述

1.1 题目介绍

在九宫格里放在 1 到 8 共 8 个数字还有一个是空格，与空格相邻的数字可以移动到空格的位置，问给定的状态最少需要几步能到达目标状态（用 0 表示空格），目标状态如图 1 所示。

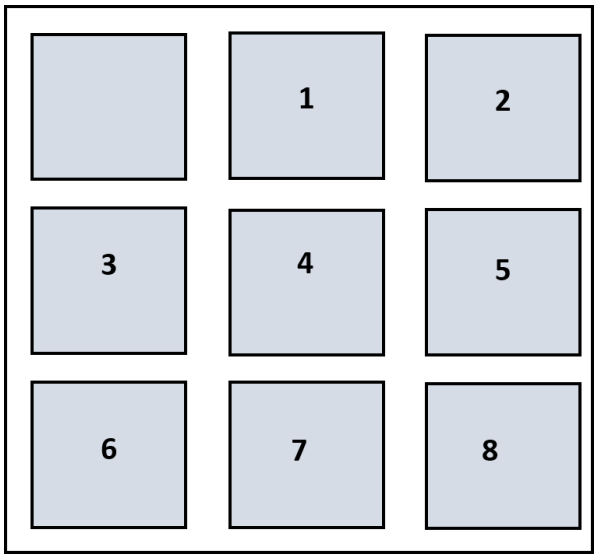

图 1 目标状态

1.2 任务说明

本次实验主要实现八数码问题的四种算法，分别为广度优先算法、深度有限算法、基于曼哈顿距离的 A*搜索算法和基于不正确数码位置的 A*搜索算法。

在框架 EightFigurePuzzlesFrameWork 中实现四个函数，在输入的回退步数和实验方法设置下能够正确输出数码移动过程、结果是否正确和移动步数。

1.3 实验环境

Visual Studio 2022

1.4 评价标准

- 搜索算法的正确性：由函数 testSearchFunction 函数验证。
- 搜索算法的搜索时间(时间复杂度)：每个搜索算法在 2 分钟内跑出结果
- 内存使用(空间复杂度)：算法内存要符合该算法的空间复杂度
- 是否是最优解

2 实验方案

2.1 关于八数码问题的分析

首先，参考书上的内容，对八数码问题进行一个“形式化”的表达：

- 状态：一个 3×3 的棋盘，在代码中的“Problem.h”中用 vector 存储。
- 初始状态：这里的初始状态在 Problem 对象实例化的时候随机生成
- 后继函数：要产生四个 Action.
- 目标测试：有专门的检测函数
- 路径耗散：这里每一步的耗散为 1

这是一种图搜索问题，每一个状态可以看成一个节点(Node)，在专门的 Node 类中定义。在问题中，数码的移动有 4 个方向，还有 failure 和 cutoff 两种状态，代码中体现在 Action 类里。而几种搜索问题要返回的结果也是一个每个元素为 Action 的 vector 数组。

实际上，不论是广度优先、深度优先还是 A^*算法，对应的伪代码都已经给出，代码实现主要是把这些伪代码进行一个复现。

2.2 广度优先解八数码问题

宽度优先算法每次拓展深度最浅的结点，这样可以把边缘组织成 FIFO 队列来实现。浅层的老节点会在深层结点之前被拓展。

2.3 深度优先解八数码问题

深度优先搜索算法是向深处拓展结点。但有的时候，可能会陷入死循环，因为会有一些分支的深度为无限深，所以会有一个限制 limit。在实验中，这个限制为 10，一旦超过这个深度，就会自动“剪枝”，进入下一个“树枝”，在代码的输出中体现为输出 CUTOFF。

2.4 A*算法解八数码问题

与之前的两个算法不同，这里的 A*算法对结点的评估结合了 g(n)和 h(n)，也就是到达此结点已经花费的代价和从该结点到目标结点所花代价。

而启发函数对 A*算法的性能也有很大的影响。这里有错位距离(misplace)和曼哈顿距离两种启发函数。前者在八数码问题中只是计算当前状态与目标态中不同的单元个数。后者则是计算曼哈顿距离$dis = |x1 - x2| + |y1 - y2|$。

3 实验结果

3.1 广度优先解八数码问题

这是运行广度优先搜索的效果截图。对于一个 18 步的 solution，耗时 2.7 秒。

```
Current State:
1 3 4
6 0 5
8 7 2
Searching tooks 2.729000 seconds.
Solution Found
Action Sequence Length: 18
Actions: RIGHT DOWN LEFT UP UP RIGHT DOWN DOWN LEFT LEFT UP RIGHT RIGHT DOWN LEFT UP UP LEFT
```

3.2 深度优先解八数码问题

这是运行 doExperiment(50, 3, 15, (searchFunc)search::dlsWrapper)函数的结果截图，这里只取了一个例子，对于一个 9 步的 Solution，深度优先算法耗时 0.091 秒。

```
Current State:
1 4 2
3 7 0
6 8 5
Searching tooks 0.091000 seconds.
Solution Found
Action Sequence Length: 9
Actions: LEFT LEFT RIGHT RIGHT DOWN LEFT UP UP LEFT
```

这是运行 testSearchFunction((searchFunc)search::dlsWrapper)函数的结果截图，可以看到，在前 10 次中，有出现 CUTOFF，这就是陷入了死循环或者较长循环后，搜索次数大于了 limit，就进行了“CUTOFF”的操作。

```
Start to test search function
Iteration 1: 1
Iteration 2: 1
Iteration 3: 1
Iteration 4: 1
Iteration 5: CUTOFF
Iteration 6: CUTOFF
Iteration 7: CUTOFF
Iteration 8: 1
Iteration 9: CUTOFF
Iteration 10: 1
```

3.3 A*算法解八数码问题

A*-Misplace：使用了 misplace 启发函数。得到一个 8 步的解耗时 0.006 秒。

```
Current State:
3 1 4
5 0 2
6 7 8
Searching tooks 0.006000 seconds.
Solution Found
Action Sequence Length: 8
Actions: LEFT UP RIGHT RIGHT DOWN LEFT UP LEFT
```

A*-manhatt：使用了 manhattan 启发函数。获得一个 16 步的解耗时 0.119 秒。这和前面的广度优先搜索比起来，已经快了很多。

```
Current State:
6 2 3
7 4 1
0 8 5
Searching tooks 0.119000 seconds.
Solution Found
Action Sequence Length: 16
Actions: UP RIGHT UP RIGHT DOWN LEFT LEFT UP RIGHT RIGHT DOWN DOWN LEFT LEFT UP UP
```

4 实验分析

经过多次测试进行时间的粗略统计，得到如下表格

算法	4 步时间约（单位：s）	8 步时间约（单位：s）	12 步时间约（单位：s）	16 步时间约（单位：s）	20 步时间约（单位：s）
广度优先	0.001	0.028	0.312	2.021	18.143
深度有限	0.001	0.133~2.5 都有	X	X	X
Misplace	0.002	0.011	0.114	0.612	2.124
Manhattan	0.001	0.013	0.045	0.25	1.184

在步数较少的情况下，有限深度优先算法和广度优先算法的耗时接近，在 10 步之后由于深度有限 CUTOFF，未能得到具体的时间，可以大致估计在高步数下，深度有限算法非常耗时。

启发式算法由于具有某些特定的规则，比广度优先和深度有限这种“暴力硬解”的方法更合理，更具有效率。

5 结论

本次实验共涉及到 4 种算法，其中广度优先和深度有限属于传统的暴力算法，效率并不高，如果实验内容不是 8-puzzle 而是 15 或者 24-puzzle，这两种算法得到解的时间会大大延长。

而启发式搜索，利用了贪心规则，其中 aStarManhattan 比 aStarMisplace 算法的贪心规则更合理，所以有更高的效率。

就本次实验要求写出的 4 种算法，aStarManhattan 具有最高的效率。