# 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

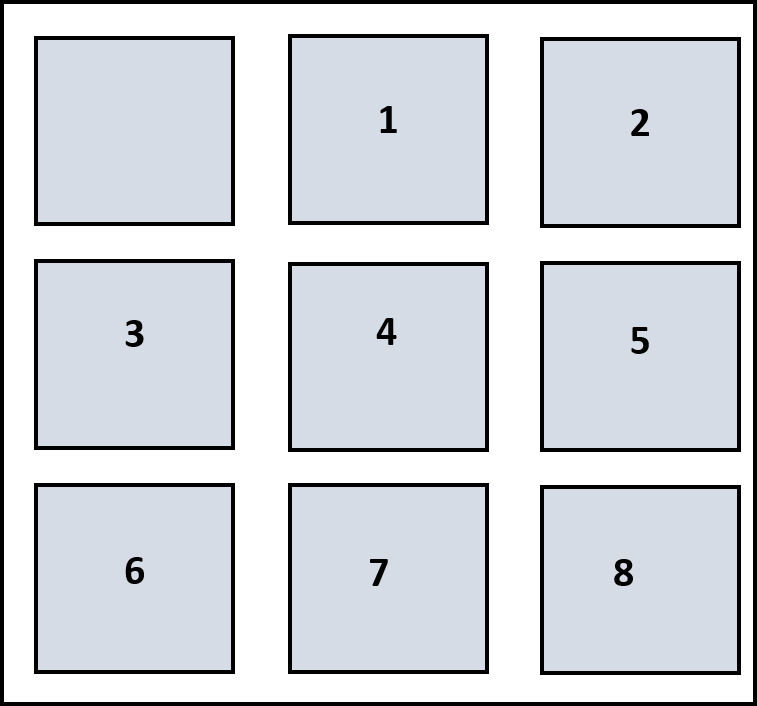
在九宫格里放在1到8共8个数字还有一个是空格，与空格相邻的数字可以移动到空格的位置，问给定的状态最少需要几步能到达目标状态（用0表示空格），目标状态如图1所示。

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。

## 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。

文本

描述已自动生成

### 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。

图形用户界面, 应用程序

描述已自动生成

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。

文本

描述已自动生成

### 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。

图形用户界面, 文本, 应用程序

描述已自动生成

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。

文本

描述已自动生成

## 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。