# 搜索作业 1 全局搜索

人工智能基础 2025 春 王乐业老师班

## 1 作业说明

1. 截止时间：5 月 11 日（周日 23:59）
2. 提交方式：教学网
3. 提交内容：所有代码文件夹、实验报告（基于 word 模板修改）。打包为.zip 后提交。

## 2 环境要求

- 【C++】11 及以上，推荐 g++ 编译器
- 【Python】3.7 及以上

## 3 常用命令

- 【C++ 编译】`g++ [文件名].cpp -o [输出文件名] --std=c++11 -O3`
- 【执行 Python 脚本】`python [文件名].py`
- 【输出重定向】`./[可执行文件名] > [输出目标文件名]`
- 【资源监控】
  - Linux/MacOS: `top`
  - Windows: `taskmgr`

## 4 样例代码

样例代码有 C++ 和 Python 两个版本，两个版本实现逻辑一致，文件层次结构一致，均提供了必要的注释辅助理解。

- 代码目录包含 4 个文件夹：

  - `interface`: 状态接口定义，包括其余文件夹下代码所依赖的状态 StateBase 接口，定义了搜索求解问题的状态表示方法。
  - `algorithm`: 包含 DFS、BFS、UCS 和启发式搜索实现，这些算法实现依赖 StateBase 接口提供的调用。
  - `problem`: $n$ 皇后和最短路径问题建模，这些状态建模实现了 StateBase 接口。
  - `utils`: 包括其余文件夹下代码依赖的工具方法与工具类。
- 代码目录包括 2 个文件：
  - `queen_bfs_dfs.cpp/.py`: 可编译/执行该文件，使用 DFS/BFS 求解八皇后问题所有解。
  - `short_path_ucs.cpp/.py`: 可编译/执行该文件，使用 UCS 求解最短路径问题的一个解。

# 5 任务描述

## 5.1 (6.5 分) $N$ 皇后问题

运行 $N$ 皇后问题的 C++/Python 代码，比较两种语言实现的深搜与广搜在皇后数为 8-15 时的时间与空间消耗，填写实验报告中的表格。

### *5.1.1 要求*

1. (2.5 分) 探究程序语言和 I/O 对程序运行时间的影响，在表格中填写：
   - 对于 C++ 语言，使用 IO (0.5 分) / 不使用 IO (0.5 分) 情况下，广度优先搜索 11/12 皇后问题时间。
   - 对于 Python 语言，重复上述过程 (0.5 + 0.5 分)。

   简要分析得到实验结果的原因 (0.5 分)。关闭 IO 的方法见提示。
2. (2.5 分) 探究深度优先和宽度优先的时间效率，在表格中填写：
   - 对于 C++ 语言，使用 BFS (0.5 分) / DFS (0.5 分) 的运行时间，表项时间单位为秒，此时需要关闭 IO。
   - 对于 Python 语言，重复上述过程 (0.5 + 0.5 分)。

   简要分析产生该实验结果的原因 (0.5 分)。
3. (1.5 分) 探究深度优先和宽度优先的空间效率，在表格中填写：
   - 对于 C++ 语言，使用 BFS (0.5 分) / DFS (0.5 分) 的空间消耗，表项空间单位为个状态，此时需要关闭 IO。

   简要分析产生该实验结果的原因 (0.5 分)。

### *5.1.2* 提示

1. 调用 DFS/BFS 的 `search` 方法时，传入参数为 `(true, false)`，因为八皇后环境的实现是逐行填子，无重复状态，是树搜索，并且也不需要记录路径。

2. 将 `search` 方法的 `show_reversed_path(last_state_of, state)` 和 `state.show()` 注释掉，即可关闭 IO。

3. 统计时间消耗，以秒为单位，如果运行时间超过 5 分钟，可提前终止进程，并在时间统计表中标注“`>300`”，空间统计表中标注“`/`”。

4. 若某种设定（程序设计语言/算法/使用 IO 情况）下某个问题的运行超过 5 分钟，则不需运行该设定下更大规模的问题求解程序，且表项填写方式同上。

5. 统计空间消耗，只需填写搜索使用的栈/队列在运行时的峰值大小，即含有的最大状态个数（C++ 代码中已经实现，会在代码运行结束时输出。注意 Python 代码并未实现相应功能，因为此处空间开销**与语言无关**）。

## 5.2 (3.5 分) 最短路径问题

利用 C++/Python 代码中给出的接口，构建有向图，求解课件第 34 页城市 Arad 到 Bucharest 的最短距离（C++/Python 代码中已经给出了地点名称的列表以及各条边的信息，以便建图使用）。

### *5.2.1* 要求

1. (1 分) 使用**一致代价搜索**（已经实现在 `uniform_cost_search` 中，注意传入的状态价值估计为 $-$( 已经走过的路程 )，即 `-state.cumulative_cost()` ）运行求解，并在报告表格中填写（**只需要写顶点编号**）：

   - (0.5 分) 各个节点进入优先队列的顺序
   - (0.5 分) 找到的路径、路径长度

2. (1 分) 使用**贪心搜索**重复上述过程，贪心的状态价值估计为 $-$( 到目标结点直线距离 )。

3. (1 分) 使用 **A* 搜索**重复上述过程，A* 的状态价值估计取 $-$( 已经走过的路程 + 到目标结点直线距离 )。

4. (0.5 分) 简要分析产生该实验结果的原因。

### *5.2.2* 提示

1. 该作业可以选择基于 C++ 版本实现或是基于 Python 版本实现，不需要在两个版本上各实现一次。但是推荐不熟悉其中一种语言的同学都尝试一遍。

2. 向 `HeuriticSearch::search` 中传入启发式函数的方式参考 `UniformCostSearch` 的实现。

3. 对于 `DirectedGraphState` 的对象 `state` 来说，`state.cumulative_cost()` 已经记录了从起点到当前点走过路径的总花费，这正是 `UniformCostSearch` 对状态估值的依据。

4. 输出路径可以参考 `utils/show_path.hpp` 中的 `show_reversed_path`。

# 6 补充材料

【Python 官方文档】`https://docs.python.org/zh-cn/3/`