上海交通大学安泰经济与管理学院

BUSS3620 人工智能导论

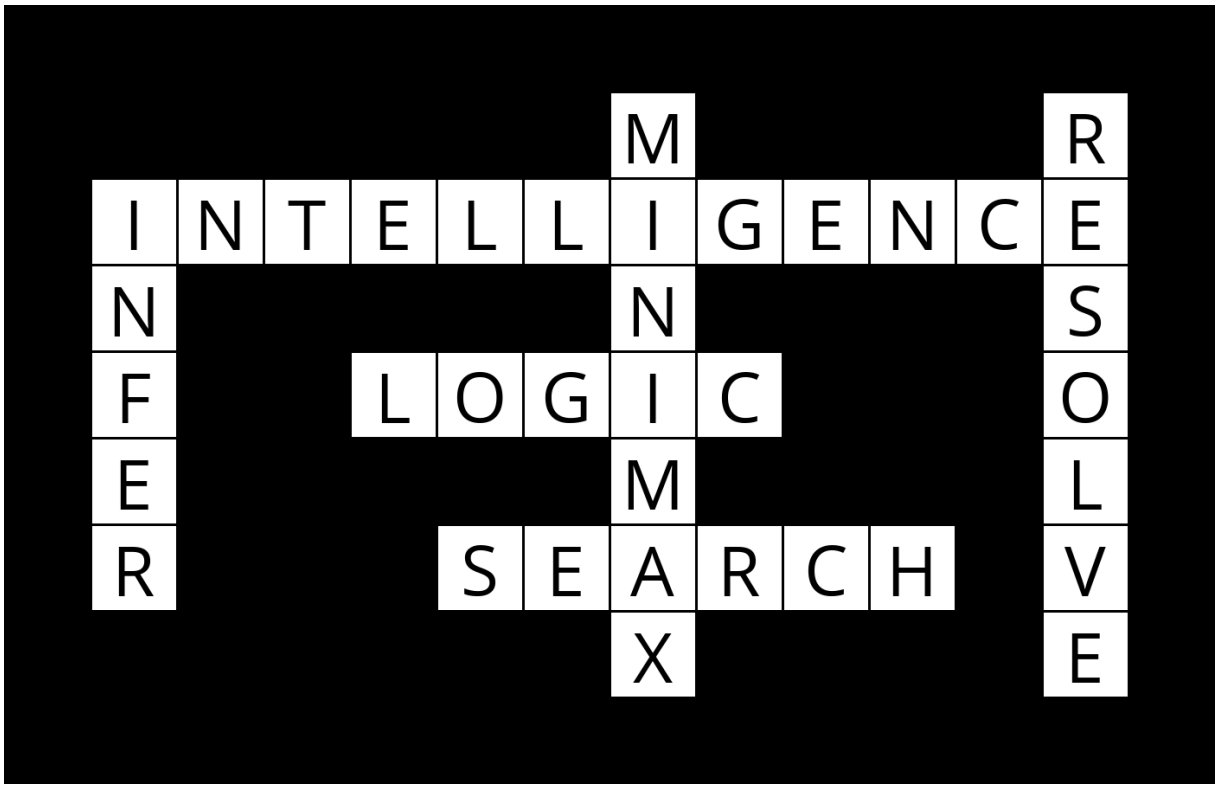
Project #4. 纵横字谜

刘佳璐 助理教授

纵横字谜

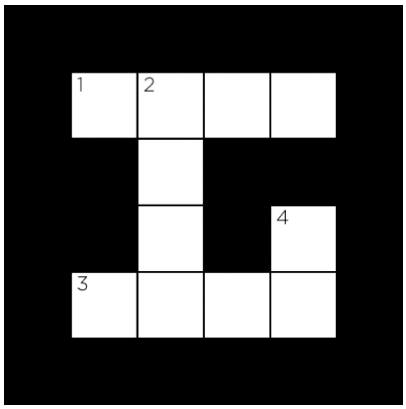
本项目将使用回溯搜索算法来创建一个可以解决纵横字谜的 AI。

```
$ python generate.py data/structure1.txt data/words1.txt output.png
```



背景介绍

你要如何制作一个可以解答纵横字谜的 AI？给定纵横字谜的结构（即网格中的哪些方格应该用字母填充）和可以使用的单词列表，纵横字谜问题就变成了应该选择哪些单词填充在垂直或水平方格里，这样我们就将这类问题转化成约束满足问题。每个方格序列都是一个变量，我们需要确定它的值（用单词定义域中的哪个单词填充该序列）。假设以下图纵横字谜结构为例：



在这个结构中，我们有四个变量，代表我们需要在这个纵横字谜里填入四个单词（每个单词在上图中用一个数字表示）。每个变量可以由四个属性定义：它开始的行（它的 `i` 值）、它开始的列（它的 `j` 值）、单词的方向（横向(`across`)或纵向(`down`)）和单词的长度。例如，变量 1 可以表示为：第 `1` 行（假设从 0 计数，从顶部开始计数）、第 `1` 列（假设从 0 计数，从左侧开始计数）、 方向 `across`、长度 `4`。

与许多约束满足问题一样，这些变量既具有一元约束，又具有二元约束。变量的一元约束由其长度给出。例如，对于变量 1，单词 `BYTE` 将满足一元约束，但单词 `BIT` 不会（它的字母长度不满足）。因此，任何不满足变量一元约束的单词都可以立即从变量的定义域中删除。

变量的二元约束由其与相邻变量的重叠位置给出。变量 1 有一个邻居：变量 2。变量 2 有两个邻居：变量 1 和变量 3。对于每对相邻变量，这些变量共享一个重叠位置：这个位置的字母必须是相同的。因此我们可以用每个变量的重叠位置的位置序号构建二元约束。例如，变量 1 和变量 2 之间的重叠位置可以表示为`(1, 0)`，这意味着变量 1 的第 `1` 个字母（从 0 计数）必须与变量 2 的第 `0` 个字母（从 0 计数）相同。因此，变量 2 和变量 3 之间的二元约束将表示为`(3, 1)`：变量 2 的第 `3` 个字母 必须与变量 3 的第 `1` 个字母相同。

对于这个问题，我们还可以添加额外的约束，即所有单词必须不同：同一个单词不应该在字谜中重复出现。

本项目的挑战是编写一个 AI 程序来解决纵横字谜：给每个变量分配一个不同的单词（来自给定的单词列表），以满足所有的一元和二元约束。

开始

- 从课程中心平台 Canvas 上下 `Week7 优化`单元中的 `week7_project.zip` 并且解压缩
- 当处于本项目文件所在的工作目录中时，在终端上运行 `pip3 install -r requirements.txt` 用来安装这次项目需要的 Python 包。

理解项目的相关文件

这个项目最主要的两个文件为：`crossword.py` 和 `generate.py`。`crossword.py` 已经写好，`generate.py` 的一些功能需要同学你来实现。有关纵横字谜的字谜结构以及单词列表存放在 `data` 文件夹中。

打开 crossword.py。这个文件定义了两个类，Variable（表示纵横字谜中的变量）和 Crossword（表示谜题本身）。

请注意，要创建一个新的 Variable 实体，我们必须提供它的 4 个属性：它所在的行 i、它所在的列 j、它的方向（Variable.ACROSS（横向）或 Variable.DOWN（纵向））以及它的长度。

类 Crossword 需要两个属性来创建新的纵横字谜：structure_file 定义谜题结构（其中_用于表示空白需填写字母的方格，任何其他字符（例如#）表示不能被填写的方格），以及 words_file 定义填写纵横字谜的单词列表（每行一个）。可以在项目目录中的 data 文件夹找到三个示例谜题结构以及三个单词列表，你也可以创建自己的示例。

特别注意，对于任何纵横字谜 crossword，我们存储以下属性：

- crossword.height 是一个整数，表示纵横字谜的高度。
- crossword.width 是一个整数，表示纵横字谜的宽度。
- crossword.structure 是一个表示谜题结构的二维列表。对于任何谜题范围内的行 i 和列 j，如果方格为空白（必须在那里填充一个字母），那么 crossword.structure[i][j]为 True，否则为 False（该单元格中不能填充任何字母）。
- crossword.words 存储填充纵横字谜时所有可以选择的单词集合。
- crossword.variables 是谜题中所有变量的集合（每个变量都是一个 Variable 对象）。
- crossword.overlaps 是一个字典，其中主键为两个变量，值为他们重叠的位置。对于任何两个不同的变量 v1 和 v2，如果这两个变量没有重叠，crossword.overlaps[v1, v2]为 None。如果这两个变量有重叠，crossword.overlaps[v1, v2]为(i, j)，即 v1 中的第 i 个字母必须与 v2 中的第 j 个字母相同。

Crossword 对象还支持一种方法 neighbors，该方法返回与给定变量重叠的所有变量。也就是说，crossword.neighbors(v1)将返回与变量 v1 相邻的所有变量的集合。

接下来，打开 generate.py。在这里，我们定义一个 CrosswordCreator 类，是我们要制作的解决字谜的 AI。当创建一个新的 CrosswordCreator 实体时，它有一个 crossword 属性，该属性应为 Crossword 实体（因此具有 Crossword 所有属性）。每个 CrosswordCreator 实体还有一个 domains 属性，他是一个字典，其中字典的主键是每个变量，值为这个变量的定义域（可能的单词）。一开始，每个变量的定义域是我们单词列表中的所有单词，但我们很快就会编写函数来缩小定义域。

一些辅助函数已经写好：print 函数会将纵横字谜的打印到终端）。save 函数将生成谜题的图像文件。letter_grid 是 print 函数与 save 函数里会用到的辅助函数：你一般不需要自己调用此函数。

最后是 solve 函数，该函数执行三件事：首先，它调用 enforce_node_consistency 函数执行字谜的节点一致性，确保变量定义域中的每个值都满足一元约束。然后，该函数调用 ac3 函数以执行弧一致性，确保满足二元约束。最后，该函数调用 backtrack 最初为空的赋值（空字典 dict()）来尝试找到字谜的解决方案。

剩下的 enforce_node_consistency 函数、ac3 函数、backtrack 函数（以及其他函数）尚未实现，需要同学你来完成。

Week7_project.zip 中还包含几个文件。asset 包含字体文件。autograde 文件夹包含测试代码的相关文件。

要求

`enforce_node_consistency` 函数

- 输入：无
- 功能：更新 `self.domains`，使得每个变量都是节点一致的，均满足其一元约束。
- 输出：无

`revise` 函数

- 输入：变量 `x`，变量 `y`
- 功能：使 `x` 对 `y` 满足弧一致性
  - 这意味着 `x` 定义域里的每个值都在 `y` 的定义域中存在一个值满足弧一致性，即删除 `x` 定义域中所有在 `y` 定义域中找不到满足二元约束的值。
    - 可以用 `self.domains[v].remove(x)`来删除变量 `v` 定义域中的一个值 `x`
    - 你可以从 `self.crossword.overlaps` 中找到对应的二元约束
  - `y` 的定义域不应该被改变
- 输出：如果对 `x` 定义域进行了修改，返回 `True`，否则返回 `False`

`ac3` 函数

- 输入：可选参数 `arcs`，表示要处理的弧的初始列表
  - 回想一下，AC3 算法需要有一个要处理的弧队列。如果 `arcs` 是 `None`，函数的 `arcs` 应为一个包含所有弧的初始队列。否则，您的算法应该处理提供的 `arcs` 队列。
  - `arcs` 中的每个弧都是一个元组`(x, y)`，表示变量 `x` 和另一个变量 `y` 的弧
- 功能：使用 AC3 算法实现弧一致性，即每个变量的定义域中的所有值都满足该变量的二元约束
  - 要实现 AC3，你每次都需要从弧队列中选择并移除一个二元约束，并使其满足弧一致。注意，每当你对一个变量的定义域进行更改时，你可能需要向队列中添加其他弧，以确保其他弧保持一致。
  - 你可以在 `ac3` 函数里调用 `revise` 函数
  - 无需在此函数中实现不能填写相同单词的这一限制（会在 `consistent` 函数中实现）
- 输出：`True` 或 `False`
  - 如果在执行弧一致性的过程中，你将一个变量的定义域中所有值都删除了，那么返回 `False`（这意味着这个约束满足问题无解，因为变量定义域为空集）。否则，返回 `True`

`assignment_complete` 函数

- 输入：一个字典 `assignment`，其中的键是 `Variable`，值是该变量赋值的单词
- 功能：检查给定 `assignment` 是否完整
  - 如果每个变量都被赋予一个值（无论该值是什么），则分配完成
- 输出：如果分配完成返回 `True`，否则返回 `False`

`consistent` 函数

- 输入：一个字典 `assignment`，注意输入的 `assignment` 可能并不完整，有一些变量可能还未进行赋值
- 功能：检查给定 `assignment` 是否一致
  - 如果 `assignment` 满足所有约束，则为一致。即所有分配的单词都是不同的，每个分配的单词的长度都和变量的长度相等，并且相邻变量之间没有冲突
- 输出：如果分配一致返回 `True`，否则返回 `False`

`order_domain_values` 函数

- 输入：变量 `var`，字典 `assignment`，注意输入的 `assignment` 可能并不完整，有一些变量可能还未进行赋值
- 功能：返回变量 `var` 包含其定义域中所有值的列表，并按照最少约束值(least-constraining value heuristics)排序，最少约束值的变量优先
  - 按照最少约束值排序，即如果赋值 `var` 某个特定值会导致排除 `n` 个相邻变量的可能选择，则应按升序对 `n` 进行排序
  - 注意如果一个变量 `var` 在 `assignment` 已被赋值，那么在计算约束值时，不应考虑该 `var`
  - 当约束值相等时，任何顺序都是可以接受的
  - 一开始时，你可以先不实现以上功能，可以返回任意顺序的定义域的值。当你成功完成 `backtrack` 函数后，可以再考虑实现以上功能
- 输出：一个列表

`select_unassigned_variable` 函数

- 输入：一个字典 `assignment`，注意输入的 `assignment` 可能并不完整，有一些变量可能还未进行赋值
- 功能：返回填字游戏中尚未赋值的单个变量，排序先根据最小剩余值启发式(minimum remaining values)，然后根据度启发式(degree heuristics)
  - 如果几个变量最小最小剩余值相等，你应该安好度启发式进行排序。如果度启发式也相等，那么任意顺序均可
  - 一开始时，你可以先不实现以上功能，可以返回任意未被赋值的变量。当你成功完成 `backtrack` 函数后，可以再考虑实现以上功能
- 输出：一个变量

`backtrack` 函数

- 输入：一个字典 `assignment`，其中的键是 `Variable`，值是该变量赋值的单词。注意输入的 `assignment` 可能并不完整，有一些变量可能还未进行赋值
- 功能：使用回溯搜索，直到所有变量到分配到一个合适的单词
  - 学有余力的同学可以引入推断（inference），你会发现算法变得更高效（正是出于这个原因，该 `ac3` 函数允许一个 `arcs` 参数，以防你想从不同的弧队列开始。）
- 输出：返回一个完整的 `assignment`，如果没有办法找到完整的 `assignment`，返回 `None`

你不应该修改 `generate.py` 中已经写好的其他部分，你不应该修改 `crossword.py` 中的任何部分。您可以使用 `numpy` 或 `pandas`，但你不应使用任何其他第三方 Python 包。

提示

- 对于 `order_domain_values` 函数和 `select_unassigned_variable` 函数，一开始可以先不考虑将输出结果按照启发式估计进行排序，这时你的程序仍然可以正常运行，只是程序可能会探索更多的分配才能找到解决方案。当你其他函数都没有问题后，你可以再考虑将这两个函数的输出结果按照启发式进行排序。
- 如果想实现排序相关的功能，可以考虑 sort 函数。
- `Crossword` 类的 `neighbors` 函数可以返回与给定变量重叠的所有变量。

测试代码

- 你可以使用代码 `pytest autograde/autograde.py --tb=no` 自行测试自己的代码是否满足要求。您需要安装 `requirements.txt` 中的 `pytest` 包。
- 请先确保你的程序能够成功运行并输出结果。请确保你的工作目录中包含 `generate.py`。