A Constraint Satisfaction Problem for Course Scheduling

Problem Description

The objective is to assign time slots and classrooms to courses while adhering to specified constraints.

The available time slots are predetermined: Mon1, Mon2, ..., Mon8, Tue1, Tue2, ..., Tue8, ..., Fri1, Fri2, ..., Fri8. These represent the first, second, ..., eighth hours of Monday, Tuesday, ..., Friday, respectively.

The constraints are below:

1. **Exclusive Classroom Assignment:** Classrooms can be assigned to only one course at a time.
2. **Capacity Compliance:** The number of students in a course must NOT exceed the capacity of the assigned classroom.
3. **Instructor Availability:** Instructors can teach only one course at a time.
4. **Consecutive Scheduling:** The time slots assigned to a course must be consecutive. For instance, Mon1 and Mon2 are consecutive time slots. There is a lunch break between the 4th and 5th hour each day. Thus, for instance, Mon4 and Mon5 are NOT consecutive time slots!
5. **Instructor Preferences Compliance:** Instructor preferences must be accommodated. For example, an instructor may wish to teach only on Tuesdays. These preferences are mandatory, NOT soft constraints.
6. **Coordination Restrictions:** Coordinated courses, which are typically taken together by the same students, must NOT be scheduled at the same time.

Your task is to assign a (starting time, classroom) pair to each course. Consequently, the domain for each variable is the Cartesian product of the time slots and classrooms.

Input Details

Courses are specified in **courses.csv**. Each row is dedicated to one course, with columns for:

- Course name (unique identifier)
- Instructor
- Number of students (the classroom must accommodate at least this number)
- Number of hours per week (indicating the consecutive hours needed for the course)

Classrooms are specified in **classrooms.csv**. Each row is dedicated to a classroom, with columns for:

- Classroom name (unique identifier)
- Capacity

Instructor preferences are specified in **preferences.csv**. Each instructor is listed at most once, with each row reflecting their preferred time slots. Columns are for:

- Instructor name (unique identifier)
- Preferred time slots (a subset of all time slots, listed as a space-separated sequence)

Coordinated courses are specified in **coordinations.csv**. Each row lists a group of coordinated courses, separated by space. Note that a course name may appear in multiple coordination groups.

Please refer to the sample files for formatting details. All files use UNIX-style line endings ('\n').

Output Requirements

Identify all viable solutions. Generate files named **1.csv**, **2.csv**, ..., with each file representing a distinct solution. Avoid file creation if no solutions exist. Sort rows by course names, and include a header "Course,Time,Classroom\n" indicating the course name, the starting time slot, and the assigned classroom. The format should mirror that of the example files.

Execution Instructions

Your program should be titled **ceng461_hw1_YourStudentId.py** and accept two **command line arguments**: the first for the input directory, the second for the output directory. For instance:

```
python3 ceng461_hw1_123456.py "/home/ersin/ceng461/hw1/problem1"
"/home/ersin/ceng461/hw1/solutions1"
```

For the above example, **courses.csv** etc. must be found in **/home/ersin/ceng461/hw1/problem1** and **1.csv** etc. must be generated in **/home/ersin/ceng461/hw1/solutions1**.

Depending on your system, you may need to use `python` instead of `python3`. Command line arguments can be retrieved via `sys.argv`.

You may assume the following:

- The input directory will contain the files as described. These will be formatted correctly and encoded in UTF-8.
- The output directory will pre-exist and be empty.

Submission Guidelines

- Ensure your code is free of syntax errors.

- Do NOT hardcode input or output directory paths; do NOT use standard input/output functions for user interaction. Employ command line arguments and file operations instead.
- Refrain from using external libraries; standard library modules are permissible.
- Submit only a single Python file. System-induced numbering due to submission updates is acceptable. Do NOT submit additional files—doing so typically indicates a misunderstanding of the requirements.
- Strive for efficiency in your code through effective search strategies (e.g., forward checking) and data structures/algorithms (e.g., utilizing sets for faster membership checking than lists). Correctness takes precedence over speed.
- Follow coding best practices, such as using clear naming conventions and modular architecture. When developing for a client, anticipate potential changes in requirements. Design your code such that a single shift in the client's needs doesn't require widespread modifications.
- Document your functions with docstrings, especially those implementing the backtracking algorithm. Include your search strategy description. For helper functions, docstrings are optional.
- Use comments to elucidate the code where necessary. General line-by-line commentary is NOT required.
- Verify your submission by downloading and running your file prior to the deadline. Post-deadline corrections are NOT permitted.
- Note that your code will be tested with different inputs. Do NOT hardcode solutions.
- Avoid plagiarism. This is an individual assignment. Plagiarized code, even if sourced from the internet or generated by AI tools, will be detected.
- Direct all inquiries to Ersin Çine via Teams well before the deadline.
- Enjoy the process.