Programming Assignment 2 — Schedule

Due to the ongoing pandemic, the educational support officers are spending many hours on scheduling the courses for each semester. In this assignment, we will use backtracking to determine how to build a schedule for an educational semester.

Description

We will try to emulate the situation of the Bachelor programs at LIACS as much as possible. There are several courses (e.g., 'Algorithms and Data structures', 'Foundations of Computer Science', . . .) and several tracks (e.g., 'Informatica', 'Bio-Informatica', 'Informatica en Economie'). Each course is offered to several tracks, and should be given at a single time slot. A schedule is constrained by a number of weekdays that the building is available, and a number of timeslots that exist per week day. Additionally, there is a finite number of lecture rooms in which the courses can be given.

For the schedule that we are going to create, there are several requirements:

- Each course should be offered exactly once in the schedule
- Two courses that are offered to the same track, can not be offered at the same time slot (since students would not be able to attend both)
- For each track, we want to limit the number of 'interval hours': hours in between lectures, on which no lecture is scheduled for that track. Each day, there may be at most one interval slot per track.
- When courses are offered for a track on a certain day, there should be at least two courses that day.

Your goal is to write a backtracking algorithm that builds a schedule that obeys these requirements. Additionally, you will write a greedy algorithm that attempts to obey these requirements.

Programming

We make the following abstraction. The schedule is implemented as a 3D string array. The first dimension spans the weekdays, the second dimension spans the slots per weekday, the last dimension spans the lecture rooms that are available. The `init` function obtains the relevant constraints (number of weekdays, number of slots per weekday, number of tracks, number of rooms available) and a `dict` containing the courses. The tracks do not have names, but are implicitly numbered (i.e., the first track is 0, . . .). The key of the dict is a string name of the course, the value is a list of integers, representing the tracks to which this course is offered.

We provided a template program. It consists of two files:

- `schedule.py`: The main assignment file.
- `schedule_test.py`: A Unit Test file.

The file `schedule.py` contains various functions that are not implemented yet (in total: 7). Note that you are not required to implement `build_schedule_backtracking`, this function makes a call to `_build_schedule_recursive`. Read the written documentation about these functions, and implement these functions. Most of these functions require less than 10 lines of code. None of the functions require more than 30 lines of code. Do not change the headers of the functions provided, except for `_build_schedule_recursive`. You are allowed to add functions yourself, if you feel that that makes it easier. Make sure to document these consistently.

The file `schedule_test.py` consists of several test functions. We provided these functions to help programming and testing the code. By running the following command, the unit tests can be executed:

```
python -m unittest schedule_test.py
```

Make sure to have the right packages installed (numpy). Feel free to add more unit tests. Do not alter the unit tests that are provided. If your program does not succeed on all unit tests that are provided, it is likely that there is still a problem in your code. Make sure that all other unit tests succeed, before submitting the code.

Additionally, also hand in a file named `test_private.py`. In here, you can create additional unit tests to verify the working of your program. Write at least 3 additional unit tests, and hand these in along with the assignment.

Also keep in mind that all unit tests should be able to run within a matter of seconds, on any computer.

Report

Write a report in LaTeX(at most 3 pages), addressing the following points / research questions:

- Introduction: describe the problem. Describe a state and action.
- Draw the state-space of a small example, e.g., using 1 weekday, 2 tracks and 4 courses. You are allowed to draw this on paper and scan the result.
- Analysis of optimal method: How much time does the optimal method take on games with varying number of courses? What is the limit that you can run within 10 minutes?
- How well does the greedy approach work? How often and how much does the answer differ from the result of the optimal algorithm? Include an adversarial test-case in your report where the greedy approach does not yield an optimal move.
- Summary and Discussion. Always end a report with some summarizing remarks.