Semester Least Time Plan Problem

By Mohammad Naji Kadri

December 13, 2018

Introduction

0.1 problem description

There are some students who want to organize their next semester in such a way that they could have more time during the day and finish their courses each day early with least time gaps possible **(break durations between two classes)**.

You need to create a program that loads semester courses from a csv file and take as input a list of courses names required by the student and print out as an output the list of courses to take next semester and their CRNs

For the simplicity of the program and to handle the problem in hand we will assume that all the courses are available for registration and each section cannot be closed.

0.2 formats

The program must follow the following formats:

- 24-hour format for time related operations %H:%M
- week days are abbreviated as follows: $M\ T\ W\ R\ F$

0.3 materials

- **fall18-19.csv** file

Design and Implementation

Implement the design structure of the classes and methods below:
(tip: test every single method you implement)

0.4 Time class

The Time class will be used to store the hours and the minutes to be used to determine the starting and ending time of each course and other time related issues.
(Note: you will need to to import the datetime module)

- **__init__()**: that takes a string of the format %H:%M and stores two variables the first variable the string itself and the second one a datetime instance of our time
- **format_time()**: a static method that takes a string as a parameter and returns datetime representation of the time
- **__sub__()**: returns the difference of time in minutes
- **__lt__()**: checks if self.time < other.time
- **__str__()**: returns the original string from the input (i.e. 9:00)
- **__repr__()**: returns a string of the following format:
  <Time: %time_string>

0.5 Day class

The Day class provides day abbreviation and day related data to access

- **days**: a static dictionary that maps the days abbreviation to their corresponding full name
- **week**: a static list that has the days abbreviation as elements
- **__init__()**: that takes a string abbreviation of the day and stores two variables the day's abbreviation and the day's full name
- **format_days()**: a static method that takes a string as a parameter and create a list of Day instances of the days found in string. i.e. string 'MWF' should return three Day instances of the days Monday, Wednesday, and Friday
- **__str__()**: returns the day's full name
- **__repr__()**: returns a string of the following format:
  <%day_abbreviation: %day_fullname>

0.6 Course class

The Course class contains all the attributes and information related to the course.

The Course class should have the following attributes:

1. course name

2. course section (i.e. L1, B1, 1, 2, CL, etc...)
3. course type (i.e. lecture, lab, etc...)
4. course CRN
5. course days (list of Day instances)
6. course starting time
7. course ending time
8. linked labs and sections (list of CRN integers)
9. link required boolean

The Course class must implement the following methods:

- **__init__()**: should have as parameters the 9 attributes (10 parameters with self) in the same order as above and store each attribute in its right format
  (Note: the parameters passed will be from the csv file containing the data so make sure to format each attribute before storing it as required)
- **__sub__()**: calculates the time gap duration between two consecutive courses
- **__eq__()**: two courses are equal or equivalent if they have the same course name and course type or they have same CRN otherwise they are not equal
- **__str__()**: returns a string of the following format:
  course name: %name, section: %section, CRN: %crn
- **__repr__()**: returns a string of the following format:
  <Course: %name - %section , CRN: %crn>

0.7 CourseManager class

This is the main portion in the program and it will contain all the necessary methods to manage and load courses, generate a new semester plan, and provide students with solutions.

The CourseManager class should implement the following methods:

- **__init__()**: takes 3 parameters, the student's name, the semester's name, and the courses csv file.
- **load_courses()**: a static method that reads a csv file and returns list of courses
- **select_courses()**: takes a list of course names as input and returns a list of all related courses and sections

- **overlap()**: takes two consecutive courses as a parameter and checks if the two courses overlap
- **conflict()**: that is given a sorted list of courses by their ending time and returns True if there is any overlap between two consecutive courses
- **daytime_break()**: that is given a sorted list of courses by their ending time and returns the total minutes of breaks (total duration of gaps) for that specific day
- **requirements_met()**: that takes a list of courses and a selected course as parameters. The requirements are:

  1. the course must be included in the semester plan
  2. one linked course must be included in the semester plan (if any)

  if there are two or more courses that are linked to our main course then the method must return False since we can only have one linked lab/section at a time
- **create_semester_plan()**: that takes a list of courses as a parameter and builds a dictionary. The keys are the days of the week (their abbreviations) and the values are lists of the courses taken at each given day
  (the lists most be ordered by their ending time)
- **contains_equiv()**: that takes a list of courses taken for the semester and returns True if there are two equivalent courses in the list
- **contains_courses()**: that takes a list of course names and list of selected courses and returns True only if there are at least one equivalent course of type 'lecture' for each one
- **valid_semester()**: as parameters the list of courses selected and the dictionary of the semester plan.
  The following conditions should be met:

  1. each course has its required linked section/lab (if it has any)
  2. there is no time conflict between courses for each single day
  3. there are no two or more equivalent courses in the same semester

  if one of the conditions isn't met then the method should return False
- **__len__()**: returns the number of available courses
- **__repr__()**: returns a string of the following format:
  %student_name 's CourseManager for %semester_name (%number_of_available_courses)
- **__repr__()**: returns a string of the following format:
  <CourseManager: %semester_name, %student_name>

0.8 solution methods implementation

Now that you have all the required classes and methods, you will next implement the solutions for our problem by using two methods: ***brute force method*** and ***greedy algorithm method***

In the CourseManager class implement the two following methods that each take a list of course names as a parameter and returns a 2-tuple (selected_courses, semester_plan):

1. **bf()**: brute force technique, can be achieved by the following algorithm:

   1) select the courses required by the student from the list of course names
   2) generate a list of all possible 2-tuples of selected courses and its semester plan by selecting different combinations of courses obtained
   3) make a list that keeps the valid semester plans and eliminate the invalid ones
   4) for each valid semester plan calculate its total break time
      (total break time is calculated by calculating the sum of the time difference between each course on each day)
   5) return the 2-tuple (semester plan) that has the minimum break time total

2. **greedy()**: greedy algorithm, greedily selecting courses by the earliest time: (we will need an extra parameter to keep track of the semester plan dictionary we are building)

   1) for each course, we select the sections and their required linked sections that have the earliest time and that doesn't conflict with the other courses that are already selected and add them to the semester plan dictionary
   2) if we have linked sections, we find the earliest time of the course lecture and then we select the list of the linked CRN courses and from it we select the one with the earliest time
   3) we keep doing this until we have added all the required courses

   note that the greedy algorithm for this problem sometimes yield the optimal solution but it is not guarenteed to always do so.
   this method can be implemented both recursively and iteratively

Testing The Program

Now that you are done with the design and implementations of the program, you will need to add some driver code to test the program to make sure our program produces the right results.

you are provided with **fall18-19.csv** file containing some sample data that will be used to test the program.

Here are two example test cases of our program:
(the order of courses in the list does not matter):

1. **Test Case 1**:

   Input:

```
c = CourseManager('Naji Kadri', 'Fall 18-19', 'fall18-19.csv')

b = c.bf(['engl203','cmps211'])

#your method/function to list courses here

print('#' * 50)

g = c.greedy(['engl203','cmps211'])

#your method/function to list courses here
```

   Expected Output:

```
1 - course name: ENGL 203, Section: 1, CRN: 312210
2 - course name: CMPS 211, Section: 1, CRN: 126800
##################################################
1 - course name: ENGL 203, Section: 1, CRN: 312210
2 - course name: CMPS 211, Section: 1, CRN: 126800
```

2. **Test Case 2**:

   Input:

```
c = CourseManager('Naji Kadri', 'Fall 18-19', 'fall18-19.csv')

b = c.bf(['math201','cmps211','engl203'])

#your method/function to list courses here

print('#' * 50)

g = c.greedy(['math201','cmps211','engl203'])
```

```
#your method/function to list courses here
```

Expected Output:

```
1 - course name: MATH 201, Section: B1, CRN: 112440
2 - course name: MATH 201, Section: 1, CRN: 112441
3 - course name: CMPS 211, Section: 2, CRN: 126805
4 - course name: ENGL 203, Section: 1, CRN: 312210
##########################################################
1 - course name: MATH 201, Section: B2, CRN: 112443
2 - course name: MATH 201, Section: 4, CRN: 112445
3 - course name: CMPS 211, Section: 2, CRN: 126805
4 - course name: ENGL 203, Section: 1, CRN: 312210
```