CS 1803

Pair Homework 6 – Greedy Scheduler (Part II)

Due: Wednesday, March 9th, 2011 before 6:00 PM
Out of 100 points

Files to submit: **1. HW6.py**

<u>This is a PAIR PROGRAMMING Assignment: Work with your partner!</u>
For pair programming assignments, you and your partner should turn in identical assignments. List both partners names at the top. Your submission must not be substantially similar to another pairs' submission. Collaboration at a reasonable level will not result in substantially similar code. Students may only collaborate with fellow students currently taking CS 1803, the TA's and the lecturer. Collaboration means talking through problems, assisting with debugging, explaining a concept, etc. You should not exchange code or write code for others.

For Help:
- TA Helpdesk – Schedule posted on class website.
- Email TA's or use T-Square Forums

Notes:
- **Don't forget to include the required comments and collaboration statement (as outlined on the course syllabus).**
- ***Do not wait until the last minute*** *to do this assignment in case you run into* problems.
- **Read the entire specifications document before starting this assignment.**

Premise

In the Greedy Scheduler (part 1) homework, you were required to read in a CSV file with information about when specific courses met, and you stored that information into a data structure so you could retrieve the information easily. Now, with this homework, we're going to use the data you previously parsed and stored to build possible class schedules (for a single day) using a "greedy" approach. For this homework, we will be focusing less on Python syntax exercises and more on algorithmic approaches to solving "real-world" problems. You are required to build a GUI for this homework. Make a single object with the following functions:

1. __init__()
2. openFileClicked()
3. insertIntoDataStruct() – Completed in part 1
4. parseLine() – Completed in part 1
5. readCSVFile() – Completed in part 1

6. doesOverlap()
7. makeScheduleMin()
8. makeScheduleMax()
9. computeNSaveClicked()

Note that our function definitions below specify the "interesting" parameters your functions will need to have, but do NOT specify the (assumed) self parameter, which you will need to add yourself because they are methods defined within a class.

File Format Information

This homework uses the same file format used in part 1. For more information, see the part 1 document.

Data Structure Information

This homework uses the same data structure used in part 1. For more information, see the part 1 document. However, this time you must modify your code which you wrote for part 1 such that the global courses variable is now a "self" variable which belongs to the object/class rather than it being a global. Additionally, your three methods (insertIntoDataStruct(), parseLine(), and readCSVFile()) which were brought over from part 1 must be modified in such a way that they will function properly inside of a class alongside the courses class instance variable.

Greedy Algorithms

Information sourced from http://en.wikipedia.org/wiki/Greedy_algorithm

A greedy algorithm is an algorithm which will attempt to find the best possible solution to a problem by making the "best choice" at each step along the way to finding the overall solution. This doesn't necessarily mean that the algorithm will produce the absolute optimum solution; in fact, it may produce the worst solution to the problem.

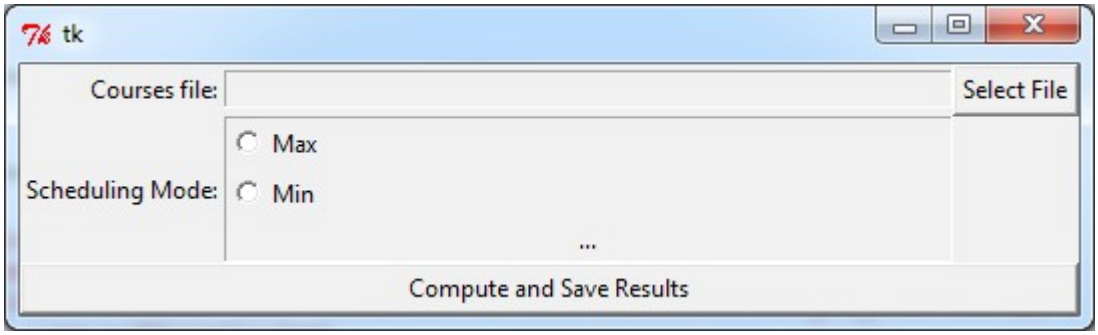
For example, let's say we're trying to make 45 cents worth of change using quarters, dimes, nickels and pennies using the least number of coins possible. The greedy solution would be to use the largest denomination of coin possible at each step to solve the problem, so we would first use a quarter, leaving 20 cents. Then, we would use a dime, leaving 10 cents. Then, we would use another dime leaving 0 cents and making a total of 45 cents. In this example, we created 45 cents using 3 coins, which is the optimum solution for the problem.

However, there are cases where this type of approach will not create the optimum solution. Imagine we had a 25-cent, a 10-cent, and a 4-cent coin. If we wanted to make 41 cents of change, the greedy algorithm would first use a 25 cent coin, leaving 16 cents. The algorithm would next use a 10-cent coin (as it is the largest denomination which will

go into 16 cents), leaving 6 cents. We can now see that the greedy algorithm will fail; there is no possible case in which we can make 6 cents with 4-cent coins, which is the only denomination we have left. However, in most cases, greedy algorithms can provide a good approximation of the optimum solution for a given input set.

Function Name: **__init__**

This method is automatically called whenever you create a new instance of your object. The __init__ method is responsible for arranging and initializing your GUI. You should create a GUI which looks like the following:



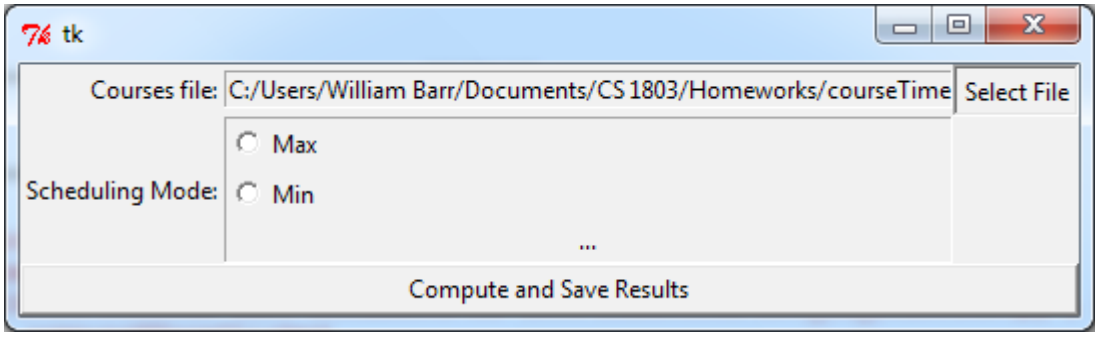
Note that the courses file Entry field is initially blank, is read only and has a width of 60.

The scheduling mode spans three rows. Next to this label is a frame which also spans three rows (Hint, it is sticky to both the east and west!). It contains two radio buttons (one with text "Max" and the other with text "Min") which are both anchored to the west. Below the two radio buttons the frame contains a label centered in the frame which initially has the text "…". Also note that the frame has a border of width 1 and the relief of the border is sunken.

The "Compute and Save Results" button spans three columns. The "Select File" button should trigger the openFileClicked() method. The "Compute and Save Results" button should trigger the computeNSaveClicked() method.

Function Name: **openFileClicked**

This method will simply launch a file open dialog box and prompt the user to select a file. If the user selects a file, the Entry Box beside the "Courses file:" label will be set to the file the user chose. If the user chooses cancel, you must set the text of this entry box to be blank.



Function Name: **doesOverlap**

Parameters:

tuple – The first of the two time pair tuples that you are checking for overlap
tuple – The second of the two time pair tuples that you are checking for overlap

Return Value:

Boolean – True if there is an overlap/conflict, false if not

Description:

This method will take in two time pair tuples and check to see whether there is any overlap between the two times. You will use this method when building your greedy schedules in order to avoid adding course times which would conflict with one another. Note, however, that if the first time pair's start time is the same as the end time of the second time pair (or vice versa), this is not a conflict (Meaning two events can be directly back to back without conflict).

Function Name: **makeScheduleMin()**

Parameters:

None (Other than the parameter all functions inside classes require)

Return Value:

A list of tuples where each tuple contains first the course name, followed by the tuple from the data structure which represents when the course is scheduled.

Description:

This method is the first of the two greedy scheduling algorithms you will write. With this particular algorithm, you will look through your list of classes and schedule them in order from the class with the least amount of possible meeting times to the class with the most number of possible meeting times. When you find the course (which you haven't already scheduled) with the least number of listed meeting times, start at the beginning of the list of times for that course and check each value sequentially until you find one which does not conflict with courses you have already scheduled. When you find this time, add it to the schedule and move onto the course with the next lowest number of meeting times. Continue this process until you have attempted to add a time for every course.

Once you are finished, return the resulting list which contains the schedule.

Note that you should not modify the Dictionary which contains all of your courses and meeting times during the course of running this method.

Function Name: **makeScheduleMax()**

Parameters:

None (Other than the parameter all functions inside classes require)

Return Value:

A list of tuples where each tuple contains first the course name, followed by the tuple from the data structure which represents when the course is scheduled.

Description:

This method is the second of the two greedy scheduling algorithms you will write. With this particular algorithm, you will look through your list of classes and schedule them in order from the class with the highest amount of possible meeting times to the class with the least number of possible meeting times. When you find the course (which you haven't already scheduled) with the greatest number of listed meeting times, start at the beginning of the list of times for that course and check each value sequentially until you find one which does not conflict with courses you have already scheduled. When you find this time, add it to the schedule and move onto the course with the next highest number of meeting times. Continue this process until you have attempted to add a time for every course.

Once you are finished, return the resulting list which contains the schedule.

Note that you should not modify the Dictionary which contains all of your courses and meeting times during the course of running this method.

Function Name: **computeNSaveClicked**

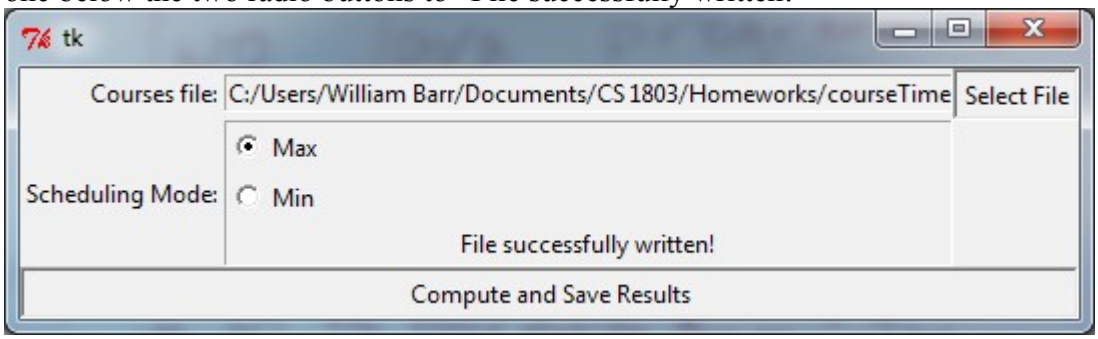
This method begins by checking to see whether there is a filename present in the "Courses File" entry box (You may assume that the user will always select a valid CSV file if they choose any file). If so, call readCSVFile and pass in the file name given in the "Courses file" entry field. Afterwards, you should prompt the user for a file to save the output to by calling the asksaveasfiledialog function. If the user selects a valid file, you should open this file and prepare it for writing. If not, you should do nothing.

This particular scheduling program has two modes: Max and Min, which are selected by the radio buttons. Next, you should see which one the user wishes to schedule; if the radio button max is selected, you should call makeScheduleMax() and prepare to write the result. If the mode is Min, you should call makeScheduleMin() and prepare to write the result. If the user selected neither radio button, you should just do nothing.

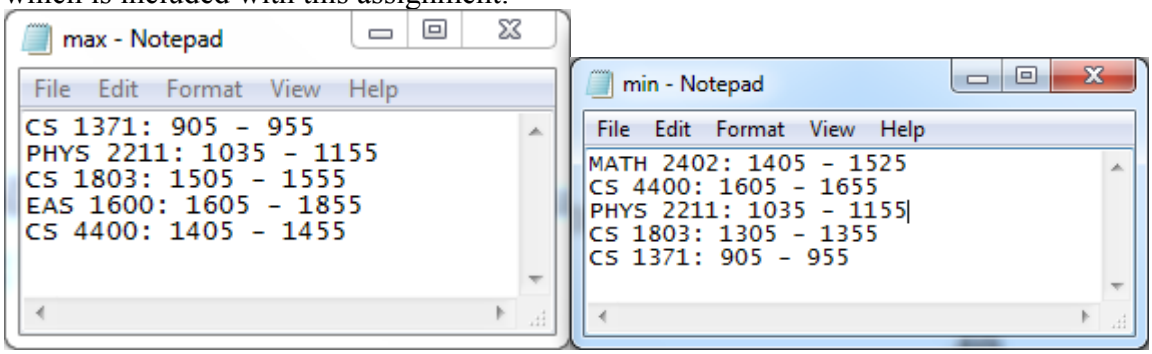
If the user completed all entry fields correctly, the program iterate over the result list that was returned by the respective scheduling function and format as follows:

Course Name: Start Time – End Time

Your program should write each course out to the output file in the above format until the entire schedule has been written to the output file, with one course per line. When the writing is complete, close the file and set the label in the "Scheduling Mode" frame (the one below the two radio buttons to "File successfully written!"



Below is the output of the min and max schedulers for the courseTimesLarge.csv file, which is included with this assignment:



You are required to use the scheduling algorithm described in the makeScheduleMin and makeScheduleMax descriptions above.

Grading:

You will earn points as follows for each piece of functionality that works correctly according to the specifications.

The GUI		**25**
GUI has all required components (includes binding)	15	
GUI has proper error checking	10	
The Part 1 Stuff*		**15**
Part 1 code updated to run inside class	10	
Data structure now stored as class instance variable	5	
doesOverlap()		**15**
Back to back courses do not overlap	5	
Correctly identifies course time conflicts/overlaps	10	
makeScheduleMax()/makeScheduleMin()		**35**
Returns data in the proper format	5	
Data returned does not have overlapping courses	5	
Both functions use the algorithm described above	5	
All courses are checked for viability in the schedule	10	
Courses are listed at most once in the schedule	10	
Miscellaneous		**10**
Data written to file is in the correct format	5	
Code is commented and is written in proper style	5	

***In addition points will be subtracted in other categories for uncorrected bugs in part 1 methods which cause errors in your part 2 functionality.**