CS 2316

Pair Homework 6 – Priority Scheduler (Part II)

Due: Thursday, June 30th, 2011 before 11:55 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 2316, 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 a possible class schedule (for a single day) using a priority 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 class 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. makeSchedule()
8. makeScheduleMin() – *OPTIONAL; EXTRA CREDIT*
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.

Additionally, there is extra credit available for this homework. The Extra Credit function makeScheduleMax may be completed for an additional 15 points of extra credit on this homework assignment.

File Format Information

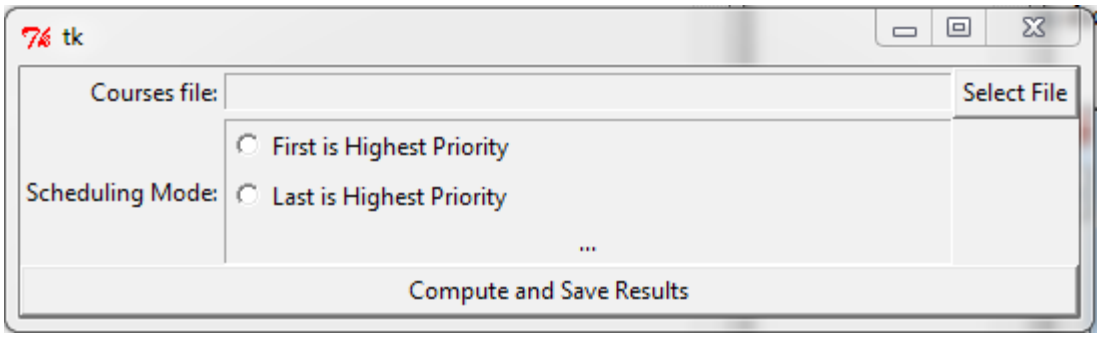
This homework uses the same file format used in part 1. For more information, see the part 1 document.

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 courses variable (the dictionary) which is passed as a parameter between functions 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. None of your functions in this homework should take the dictionary as a parameter (or return it), since you should refer to the courses dictionary as an instance variable.

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:



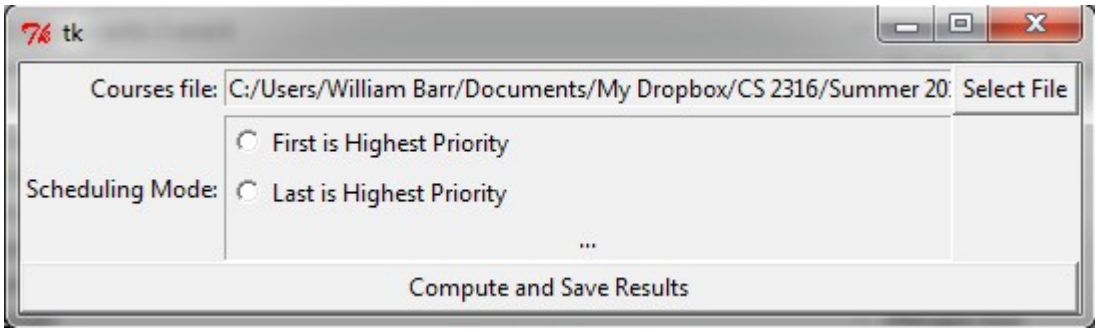
Note that the courses file Entry field is initially blank, is read only and has a width of 60.

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, neither of which are initially selected (one with text "First is Highest Priority" and the other with text "Last is Highest Priority") 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 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: **makeSchedule()**

Parameters:

highPriority – A string that is either "first" (Representing that the first course encountered in the priority file is the highest priority) or "last" representing that the last course encountered in the priority file is the highest priority)

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 will open a text file (located in the same directory as your python file) named "prioritylist.txt". This text file will contain a list of course names in a specific order from the course times file you read in. Once you have finished reading in these course names, you will schedule the first time for the highest priority course from the list of times for that course (either the first course in the text file or the last, depending on the highPriority parameter). The format of the scheduled classes that you return should be a list of tuples where each tuple contains the course name, followed by the tuple from the courses data structure that represents the timespan of the section you scheduled (i.e. *("CS 2316", (1000, 1145))*). On the next iteration, move to the next course to schedule (either the second course name or the second to last course name, depending on highPriority), and schedule the first valid course time from that course's list of times. A valid course time is defined as one that does not overlap with a course time that has already been scheduled. Repeat this process until you have scheduled all courses from the text file which can be scheduled (Note that each course should be scheduled a maximum of one time). It is possible that you may not be able to schedule a particular course as it has no valid times to schedule (i.e. if all available course times are already used by higher-priority courses). If this is the case, simply do not include that class in the schedule you return.

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: **makeScheduleMin() – OPTIONAL**

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.

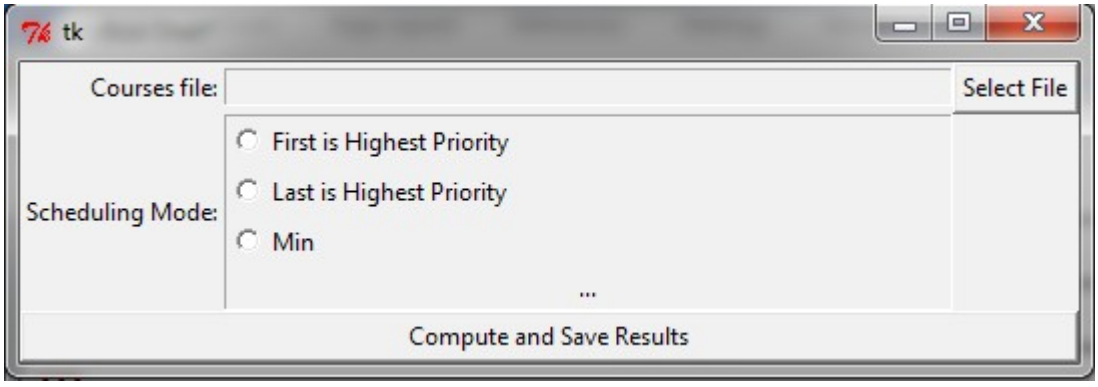
Description:

This method will implement a greedy scheduling algorithm. 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.

If you implement this method, you should modify your GUI to add an additional radio button below the first two, with the text "Min". Note that the "min" scheduler does not use the priorityList.txt file in any way!



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

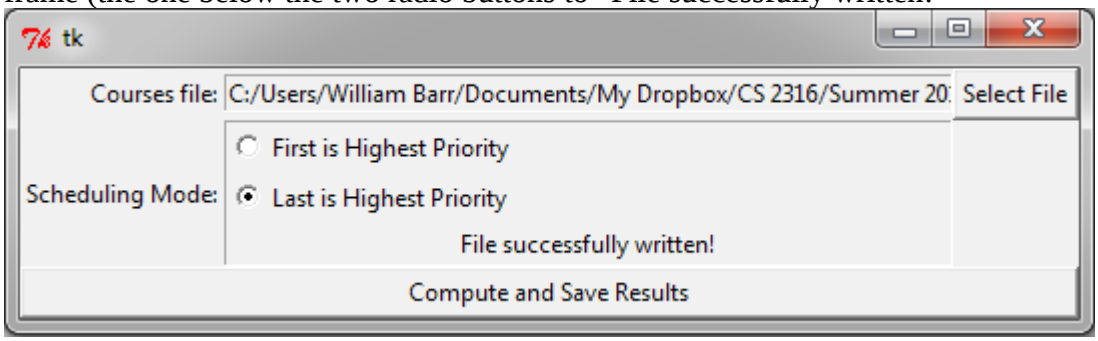
“Courses file” entry field. Afterwards, you should prompt the user for a file to save the output to by calling the asksaveasfiledialog function. If the user selects a valid file (i.e. doesn’t push cancel), you should open this file and prepare it for writing. If not, you should do nothing.

This particular scheduling program has two (or three, if you do the extra credit) modes: First is Highest Priority and Last is Highest Priority (Along with Min if you do the extra credit), which are selected by the radio buttons. Next, you should see which one the user wishes to schedule; if the radio button “First is Highest Priority” is selected, you should call makeSchedule() with the parameter being “first” and prepare to write the result. If the mode is “Last is Highest Priority”, you should call makeSchedule () with the parameter being “last” and prepare to write the result. If you did the extra credit and the user selects “Min”, you should call makeScheduleMin() and prepare to write the result. If the user selected none of the radio button, you should just do nothing.

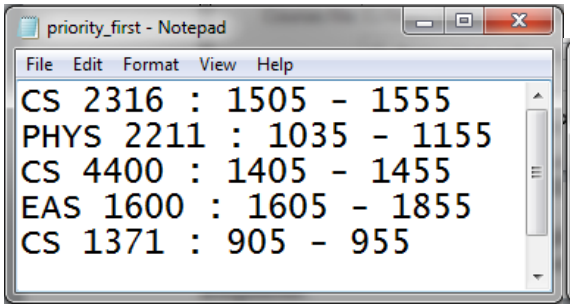
If the user completed all entry fields correctly, the program should iterate over the result list that was returned by the respective scheduling function and format each item as follows:

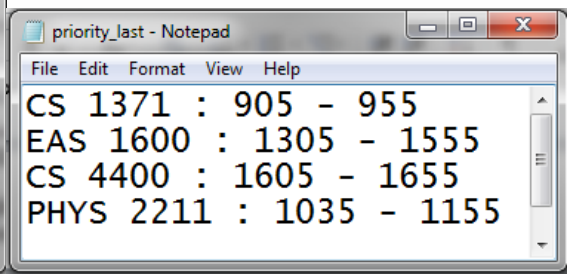
Course Name: Start Time – End Time

Your program should write each course out to the output text 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!”

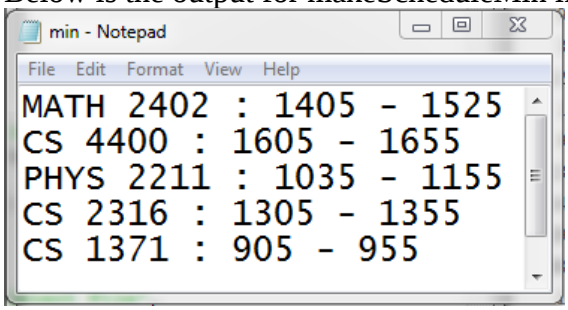


Below is the output of the First is Highest Priority and Last is Highest Priority schedulers for the courseTimesLarge.csv and prioritylist.txt files, which are included with this assignment:





Below is the output for makeScheduleMin if you do the extra credit:



You are required to use the exact scheduling algorithm described in the makeSchedule (and makeScheduleMin, if you do the extra credit) descriptions above. We suggest you make up your own priority file and add courses and times to courseTimesLarge.csv to further test your program.

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	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	
makeSchedule		**35**
Returns data in the proper format	5	
Priority List file read in and closed	5	
Data returned does not have overlapping courses	5	
Function correctly implements the algorithm	10	
All courses are checked for viability in the schedule	5	
Courses are listed at most once in the schedule	5	
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.**