EECS 214: Data Structures
9/6/2014
Team Juliett Project Report

For the EECS 214 group project, Team Juliett designed a course scheduler for Northwestern students. As a group, we determined that sorting out a schedule on CAESAR can be tedious, limiting, and frustrating. The Brutus Course Planner is not an alternative to CAESAR, but rather a supplementary scheduling device. Any user may input their name, residence, and time preferences, along with courses that they want to take next quarter. Brutus considers all possible schedules that can be created from the different combinations of non-overlapping courses, and returns one that aligns with the user's preferences. Brutus is not perfect and there is room for improvement, but the scheduler has been tested and runs well.

How the Program Works

The Front End

This is how you, the user, will interact with Brutus. Visit **http://juliett-nu.herokuapp.com/** to examine the course scheduler's features and see how it works. With regards to input, the user enters some information: his/her name, his location on campus (if minimizing walking distance is important), and preferred start and end times. Next, the user "searches" for courses by beginning to type them in the course box. A menu with course offerings will drop down once the system recognizes the course/subject that is being searched for. The best way to use this box is to enter a course's subject tag (EECS, PHIL, MATH, SPANISH, e.g.) and the first number of its level. For example, if the user wants to select EECS 214, it is best to type EECS 2… and select from the drop down menu the proper course. Then the user may select whether or not this course is "mandatory" or "optional" for his/her schedule next quarter.

The Back End

When the user presses 'submit', all of the user's preferences are passed to the backend, which creates a new user Bob. The back end is also passed integer arrays with the courseIDs of the submitted courses as two different arrays, one for mandatory and another for optional. It searches a text file database of course listings (the database currently being accessed is for Fall 2013) in order to create Course object arrays. These arrays are used to construct an 'unchecked' Schedule object. The ScheduleChecker class uses the unchecked schedule object as input in order to generate an array of schedules (using the resolveConflicts() method) that contain in them an array of four courses with no time conflicts. This array is passed, along with Bob and his course preferences, to the ScheduleGenerator class which calls self contained methods to rank each schedule and return the best schedule. The final method getSchedule returns an int array with 4 (or less) course IDs which the front end parses to construct the output.

The Front End

Once the back end has done its work, and the four courseIDs are passed to the front end, the front end utilizes the same text file database to gather all the relevant course information. The courses and their information are subsequently arranged into a weekly schedule format. The user

is able to view the time specifics of the generated schedule, and also may mouse over the course listings for more information, such as the course's topic, instructor, section, and location.

Classes Involved

The front end is designed in HTML, but all of the back end is designed in java. The team_juliett_processor package contains 6 classes. Here are the classes and how they work:
User - a user contains user start/end time preferences, an integer representation of location, and integer arrays of courseIDs, all provided from the front end. A user is constructed to access preferences, and courseIDs are parsed in order to construct Course objects.

Course - a Course object contains all of the relevant information any regular school course should have: name, days of the week that it meets, start/end times, location, and an ID. Course locations are split between buildings North of Kellogg and South of Kellogg

Schedule - the processor utilizes the object Schedule for two main purposes. A Schedule is initially constructed with two arrays of courses, mandatory and optional. Schedule objects are manipulated in ScheduleChecker and "reconstructed" to represent what a user's actual schedule will look like. That is, this Schedule contains the Course array all_courses which holds four courses without time conflicts; this is the schedule type that will be rated by the ScheduleGenerator class.

ScheduleChecker - this is where the processor gets to work. ScheduleChecker stores the first of the two Schedule objects referenced above, and runs the self contained resolveConflicts() function which returns an array of Schedule objects (this time they are the second of the two mentioned above). ScheduleChecker creates an ArrayList<ArrayList<Courses>> (array list of array lists of courses) representing all the possible combinations of courses. It removes all members of this list with time conflicts, and then trims the list so that it only contains lists of courses with four members (or less, if the user does not provide enough). These course lists are passed to a Schedule constructor and to-be-rated schedules are created.

ScheduleGenerator - the ScheduleGenerator class takes as input the array of Schedules generated in ScheduleChecker, along with Bob and the two course arrays constructed by the first type in the Schedule class. It uses this information to rank each schedule and give it a scheduleScore. The contained getBestSchedule() method is then called to return the best schedule.

Team_Juliett_processor - this is the main class. It utilizes all of the classes above, and returns an integer array of courseIDs to the front end, in a process as follows:

```
schedule unChecked = new schedule(bobsMcourses,bobsOcourses);
ScheduleChecker a = new ScheduleChecker(unChecked);
schedule[] validSchedules = a.resolveConflicts();
ScheduleGenerator b = new ScheduleGenerator(validSchedules, bob,
                                    bobsMandatoryCourses, bobsOptionalCourses);
schedule finalSchedule = b.getBestSchedule();
```

Room for Improvement with Future Design

We are currently happy with the functionality of our program. Team Juliett believes that the processor is considering the most important parameters for schedule ranking. Most important is the quantity of "mandatory" courses a given generated schedule contains. Next is whether or not the schedule fits your time preferences. Third is how much walking distance will be done on a given day. Still, there are some things that could add value to Brutus (both in design and implementation) that our Team either considered during the planning stages and did not execute on, or realized later on in the development of the project.

First, the functionality of our program would benefit if the output contained not one optimal schedule, but 3-5. They could be loaded on the same web page in scrolling list format or tabbed format. Having more options gives the user more schedules to choose from and examine. He or she may get a better feel for how the ranking system works, and also is given the flexibility of viewing different schedules that work. It is very possible that a user would prefer the arrangement of the third best schedule (by Brutus rankings) compared to the first.

Second, we would like to implement a warning message system that provided the user with certain messages pertinent to a schedule output. For example, if the "best" created schedule starts or ends outside the user's preferred schedule times, it would be beneficial if a message such as "hey, this schedule starts a little early on Tuesdays and Thursdays!" accompanied a schedule in which that was the case. Other message ideas include "careful, you may not be able to eat lunch on Wednesdays!" or "check it out! no class on Fridays!", among others. This does not seem like something too difficult to implement, but would require more compatibility between the front and back end.

Finally, there are more parameters to consider when creating and ranking schedules. Perhaps a student works part-time and his/her job requires a full day of work on Thursday. This eliminates the possibility of class on thursdays from 9-5. There is not currently a way for Brutus to handle this scenario. Another example is a student who is looking for a WCAS distribution requirement in Area VI, and he or she has not looked at what is offered in the coming quarter. It would be helpful if Brutus could return schedules with courses that fill a certain unspecified requirement of this type. We also believe Northwestern students would benefit if there was a way to access and integrate CTEC ratings, so that when the schedule is being ranked, "highest CTEC score" is something to consider.

Our team worked well together and enjoyed the design of our Brutus processor. At the same time, we have only scratched the surface regarding a truly robust scheduling device for Northwestern Students, and there is definite possibility for future improvement.