CSCI 3901 Winter 2021

Assignment 1

Due date: 23:59 AST Friday, January 29, 2021 in Brightspace

Problem 1

Goal

Get practice in decomposing a problem, creating a design for a program, and implementing and testing a program.

Background

Recommender systems use past behaviours of others to give you suggestions on what has worked for others in the past, based on your own current context. Amazon books, for example, looks at the set of books that you have already purchased and compares your reading list with the reading list of others. It then aggregates the purchases of others who also bought many of the same books as you and recommends to you those books from the aggregate that you haven't already purchased. Typically, you don't make a recommendation unless there is enough data to suggest that it's a meaningful recommendation.

In this assignment, we will do a simplified version of a recommender system. In our case, you will be recommending courses to take at the university based on what other students have taken.

Problem

Design a class `CourseSelector` that collects data on which courses students have taken, and then allows a user to make queries about the data. You will be provided with a `main` program that will then let a user invoke the operations on the data.

Your class will read data from a file, then allow a user to request the following three queries:

- Recommend n courses based on a given list of courses
- Print to a file the number of students who have taken any pair of courses
- Print to screen the number of students who have taken any pair of a given list of courses

Specifically, you will implement the following class:

```
class CourseSelector {
        public int read(String filename) { /* IMPLEMENT */ }

        public ArrayList<String> recommend(String taken,
                                           int support,
                                           int numRec)
        { /* IMPLEMENT */ }

        public boolean showCommon(String courses) { /* IMPLEMENT */ }

        public boolean showCommonAll(String filename) { /* IMPLEMENT */ }
}
```

Descriptions of the methods follow below.

`int read(String filename)` The `read` method reads the contents of a file named `filename` to the object. Each line represents the courses a student has taken. The method returns the number of rows read in. The data should replace any data previously read into the class. Individual courses in a line are separated by spaces.

`ArrayList<String> recommend(String taken, int support, int numRec)` This method returns a list of `numRec` course recommendations given a list (`taken`) of courses already taken, separated by spaces. Only report courses if the recommendations are supported by at least `support` other students.

More specifically, the `recommend` method should

1. Find all of the students who have taken all of the same courses listed in `taken`.
2. If there are at least `support` students who have taken all the courses, then
   (a) Find all of the courses these students have taken, not including the courses listed in `taken`.
   (b) For each of these courses, count the number of times they appear
   (c) Report back the first `numRec` most taken courses

If you don't have enough students to make a recommendation, then return `null`.

If you do have recommendations to return, then the returned `ArrayList` should have at most `numRec` in it. The courses should be provided in descending order of frequency. Two courses with the same frequency can be in any order.

There is one situation where the `ArrayList` can have more entries: when you reach `numRec` entries and there are other courses with the same frequency that you haven't reported. Rather than choose between which of these last courses to return, report all of the ones with that frequency.

For example, suppose that the outcome of computation 2c is the following list of courses:

Course	Number of students who have taken it
CSCI1100	12
CSCI2110	10
CSCI2112	8
CSCI2134	8
CSCI3110	5
CSCI3120	4

- For 2 recommendations, return CSCI1100 and CSCI2110.
- For 3 recommendations, return CSCI1100, CSCI2110, CSCI2112, and CSCI2134 (since the last two both have the same frequency).
- For 5 recommendations, return CSCI1100, CSCI2110, CSCI2112, CSCI2134, and CSCI3110.

`boolean showCommon(String courses)` This method prints to the screen a 2d array with one row and column for each course in `courses` in the order in which they appear. The value in each entry is the number of students who have taken both courses. Return `False` if any errors were encountered and `True` otherwise.

`boolean showCommonAll(String filename)` This method prints to the file `filename` a 2d array with one row and column for every course that has been taken by any student, in order of course name. The value in each entry is the number of students who have taken both courses. Return `False` if any errors were encountered and `True` otherwise.

Both methods with this method name compute a pairwise popularity matrix. Pairs of courses that are both taken frequently shouldn't be scheduled at the same time, so we want to know the pairs of courses that are both taken often.

The basic logic is as follows:

1. Create a 2d array with one row and one column for each course that we are given.
2. Consider the courses of each student in our system.
   (a) Find the rows/column number for each of the courses.
   (b) For each pair of courses, add 1 to the entry in the 2d array for the pair. Do not pair a course with itself.
3. Print the resulting matrix.

When you print the matrix, you do not print the column names. For each row of the matrix, print the course name and then the integer from each column. Each of these bits to print will be separated by a tab character.

For example, suppose that we have entries for the following students in our system:

CSCI1110 CSCI2110 CSCI2122
CSCI2110 CSCI2112 CSCI2122 CSCI2134
CSCI1110 CSCI2134 CSCI3171
CSCI2141

If we asked for a matrix for all courses above, we would get the following output

CSCI1110	0	1	0	1	1	0	1
CSCI2110	1	0	1	2	1	0	0
CSCI2112	0	1	0	1	1	0	0
CSCI2122	1	2	1	0	1	0	0
CSCI2134	1	1	1	1	0	0	1
CSCI2141	0	0	0	0	0	0	0
CSCI3171	1	0	0	0	1	0	0

How to submit

Submit the `.java` file containing your `CourseSelector` class and any other supporting `.java` files to Brightspace.

Inputs

An input data file for your class will have one row for each student. One row consists of a set of course numbers (8 characters each), separated by spaces.

Each course number in the file will be an alphanumeric string like CSCI3901.

You can assume that no student will have more than 10 courses provided in the file. The courses provided for a student are not necessarily provided alphabetically.

Example:

CSCI3901 CSCI5100
CSCI1110 CSCI2112 CSCI2110 CSCI2134 CSCI2141

Assumptions

You may assume that

- Each line in the input file will have at most 10 courses.
- There are no spaces included in the course names in the input file or in the parameters provided to your methods.
- File names will contain the full path to the file, including any file name extensions.

Constraints

- Write your solution in Java. You are allowed to use data structures from the Java Collection Framework.
- Course numbers are case insensitive.
- If in doubt for testing, I will be running your program on `timberlea.cs.dal.ca`. Correct operation of your program shouldn't rely on any packages that aren't available on that system.

Notes

- You will be provided with a `main` class on the course web page that will give you an interface through which to invoke your class.
- It is possible (and acceptable) to do this assignment with basic Java data types and `ArrayLists`. You are allowed to use or create more complex data types if you want.
- The functionality of your assignment will be assessed based on the results of test cases. Consequently, code that doesn't compile will not get marks for functionality.
- You are permitted to create more than one class in support of your work.
- I recommend thinking about how you want to store your data first. Consider some of the built-in data types that do not worry about the size of the data. Next, design, code, and test functionality one method at a time. The `showCommon`/`showCommonAll` methods are likely easier to start with than the `recommend` method.
- Look for common tasks that you can re-use between methods. Break these out as new private methods.
- Document your code. **If I hide the Java code, I should still get a reasonable sense of your method from the sequence of comments.**
- When testing, your methods will be invoked with error conditions. You are expected to handle these errors.
- Review the test cases to identify special cases that you will want to consider in your work.
- Your class should never end by producing a Java error stack trace.
- The marking scheme has no marks for time or space efficiency.
- Use the number of marks for the methods and the set of test cases as guides for how much time to devote to each method.
- Recognize when you are spending time with no progress and ask for help before you find yourself spending hours with no progress to show for the time.

Marking scheme (out of 30)

- Code documentation – 3 marks
- Program organization, clarity, modularity, style – 4 marks
- Ability to process file contents (read, write) – 5 marks
- Working showCommon/showCommonAll methods – 8 marks
- Working recommend method – 10 marks

The majority of the functional testing will be done with an automated script, so stick to the input and output formats.

Test cases

Input Validation (usually looking for bad data and ensure you don't crash)

- Null value passed as filename to read or showCommonAll
- Empty string passed as filename to read or showCommonAll
- Courses separated by more than one space in read, recommend, showCommon
- Recommend method
  - `taken` value is null
  - `taken` value is an empty string
  - `support` value of -1
  - `numRec` value of -1
- showCommon
  - `courses` parameter is null
  - `courses` parameter is an empty string

Boundary Cases (includes special or edge cases to consider)

- Read method
  - File does not exist
  - Empty file
  - 1 line in the file
  - Student has just 1 course
  - Student has 10 courses
- Recommend method
  - `taken` parameter only has 1 course
  - `support` parameter is 0
  - `support` parameter is 1
  - `support` parameter is more than the number of students in the object
  - `numRec` parameter is 0
  - `numRec` parameter is 1
- showCommon method
  - 1 course in `taken` list
  - There are no overlaps between any of the courses taken (each student only has 1 course)
  - Some pair of courses is only taken together once
  - Make the matrix from a single student
- showCommonAll method
  - 1 student in the object
    - With 1 course
    - With 10 courses

  - There are no overlaps between any of the courses taken (each student only has 1 course)
  - Some pair of courses is only taken together once
  - Make the matrix from a single student

Control Flow (think of it as "normal" cases for now)

- Read method
  - Many student lines in the file
  - Student has 4 courses (or a non-boundary number of courses)
- Recommend method
  - `taken` parameter has 2 courses
  - `taken` parameter has all courses in it
  - `taken` parameter contains a course that no student has taken
  - `support` parameter is a reasonable positive integer
  - `numRec` parameter is a reasonable positive integer
  - Number of courses to recommend is below `numRec`
  - Number of courses to recommend is exactly `numRec`
  - Number of courses to recommend is greater than `numRec` and there are no ties at the `numRec` boundary
  - Number of courses to recommend is greater than `numRec` and there is a tie of courses to report at the `numRec` boundary
- showCommon method
  - `courses` parameter comes in with 2 courses
  - `courses` parameter comes in with several courses
  - `courses` parameter comes in with a course that no student has taken
  - `courses` parameter has courses in a different order than what they were when read in and also not sorted alphabetically
  - Called with many students' courses read into the object
- showCommonAll method
  - call with many students loaded into the object, using a varying number of courses per student
  - courses were reported for students non-alphabetically

Data Flow (think of it as calling things in an unexpected order for now)

- Read data into a new object
- Read data into an object that already has data in it
- Call showCommon or showCommonAll with no students read into the object
- Call recommend, showCommon, or showCommonAll more than once on the same object