Programming Assignment #1

Programming assignments are to be done individually. You may discuss the problem and general concepts with other students, but there should be no sharing of code. You may not submit code other than that which you write yourself or is provided with the assignment. This restriction specifically prohibits downloading code from the Internet. If any code you submit is in violation of this policy, you will receive no credit for the entire assignment.

The goals of this lab are:

- Familiarize you with programming in Java
- Show an application of the stable matching problem
- Understand the difference between the two optimal stable matchings.

Problem Description

In this project, you will implement a variation of the stable marriage problem adapted from the textbook Chapter 1, Exercise 4, and write a small report. We have provided Java code skeletons that you will fill in with your own solution. Please read through this document and the documentation in the starter code thoroughly before beginning.

There are 2600 Mattress Firm locations in the United States. Each location belongs to a district, and each district shares the same group of sales people to schedule for duty.

The Austin Mattress Firm district is trying to set up next week's assignment of sales people to store locations. Each sales person has ranking on the locations in order of preference, and each location has a ranking on the sales people in order of their sales.

A group of sales people can work the floor of a single Mattress Firm store, but each sales person can only be assigned to one store location. Assume there are more sales people than positions and some people may not be scheduled this week. Your job is to find a stable assignment of sales people to Mattress Firm locations.

We say that an assignment of sales people to stores is *stable* if neither of the following situations arises:

- First type of instability: There are sales people s and s', and a Mattress Firm store h, such that
  - s is assigned to h, and
  - s' is assigned to no store, and
  - h prefers s' to s
- Second type of instability: There are sales people s and s', and stores h and h', so that

- s is assigned to h, and
- s' is assigned to h', and
- h prefers s' to s, and
- s' prefers h to h'.

So we basically have the Stable Matching Problem as presented in class, except that (i) store locations can have and generally want more than one sales person, and (ii) there is a surplus of sales people. There are several parts to the problem.

Part 1: Write a report [20 points]
Write a short report that includes the following information:

(a) Give an algorithm in pseudocode (only psuedocode!) to find a stable assignment that is **location** optimal. *Hint: it should be very similar to the Gale-Shapley algorithm, with locations taking the role of the men, and sales employees of the women.*

(b) Using m as number of locations and n as number of sales employees, give the runtime complexity of your algorithm in (a) in Big O notation and explain why. **Note: Full credit will be given to solutions that have a complexity of O(mn)**.

(c) Give an algorithm in pseudocode (only psuedocode!) to find a stable assignment that is **employee** optimal *Hint: it should be very similar to the Gale-Shapley algorithm, with employees taking the role of the men, and locations of the women.*

(d) Give the runtime complexity of your algorithm in (c) in Big O notation and explain why. **Note : Try to make your algorithm as efficient as you can, but you may get full credit even if it is not O(mn) as long as you clearly explain your running time and the difficulty of optimizing it further.**

In the following two sections you will implement code for a stability checking method and an efficient algorithm to find both employee and location solutions. **Note : For the programming assignment, you don't need to submit a proof that your algorithm returns a stable matching, or of location or employee optimality.**

Part 2: Implement a Verifier that Checks if a Matching is Stable [20 points]
For this part of the assignment, you are to implement a function that verifies whether or not a given matching is stable and find the employee optimal one. Inside `Program1.java` is the placeholder for a verify function called `isStableMatching()`. Implement this function to check if your other matchings are stable.
Of the files we have provided, please only modify `Problem1.java`, so that your solution remains compatible with ours. However, feel free to add any additional Java files (of your own authorship) as you see fit.

Part 3: Implement an Efficient Algorithm [60 points]
Implement the efficient algorithm you devised in your report for employee optimal and location optimal solutions. Again, you are provided several files to work with. Implement the function that

yields a employee optimal solution `stableMarriageGaleShapley_employeeoptimal()` and location optimal solution `stableMarriageGaleShapley_locationoptimal()` inside of `Program1.java`. Of the files we have provided, please only modify `Problem1.java`, so that your solution remains compatible with ours. However, feel free to add any additional Java files (of your own authorship) as you see fit.

Instructions

- Download and import the code into your favorite development environment. We will be grading in Java 1.8. Therefore, we recommend you use Java 1.8 and NOT other versions of Java, as we can not guarantee that other versions of Java will be compatible with our grading scripts. **It is YOUR responsibility to ensure that your solution compiles with Java 1.8.** If you have doubts, email a TA or post your question on Piazza.
- If you do not know how to download Java or are having trouble choosing and running an IDE, email a TA, post your question on Piazza or visit the TAs during Office Hours.
- There are several `.java` files, but you only need to make modifications to `Program1.java`. **Do not modify the other files.** However, you may add additional source files in your solution if you so desire. There is a lot of starter code; carefully study the code provided for you, and ensure that you understand it before starting to code your solution. The set of provided files should compile and run successfully before you modify them.
- The main data structure for a matching is defined and documented in `Matching.java`. A Matching object includes:
  - **m**: The number of locations
  - **n**: Number of students
  - **location_preference**: An ArrayList of ArrayLists containing each of the location's preferences of students, in order from most preferred to least preferred. The locations are in order from 0 to $m-1$. Each location has an ArrayList that ranks its preferences of students who are identified by numbers 0 through $n-1$.
  - **employee_preference**: An ArrayList of ArrayLists containing each of the student's preferences for locations, in order from most preferred to least preferred. The students are in order from 0 to $n-1$. Each student has an ArrayList that ranks its preferences of locations who are identified by numbers 0 to $m-1$.
  - **location_slots**: An ArrayList that specifies how many slots each location has. The index of the value corresponds to which location it represents.
  - **employee_matching**: An ArrayList to hold the final matching. This ArrayList (should) hold the number of the location each student is assigned to. This field will be empty in the `Matching` which is passed to your functions. The results of your algorithm should be stored in this field either by calling `setEmployeeMatching(<your_solution>)` or constructing a `new Matching(marriage, <your_solution>)`, where `marriage` is the Matching we pass into the function. The index of this ArrayList corresponds to each student. The value at that index indicates to which location he/she is matched. A

value of -1 at that index indicates that the student is not matched up. For example, if student 0 is matched to location 55, student 1 is unmatched, and student 2 is matched to location 3, the ArrayList should contain {55, -1, 3}.

- You must implement the methods

  ```
  isStableMatching()
  stableMarriageGaleShapley_employeeoptimal()
  stableMarriageGaleShapley_locationoptimal()
  ```

  in the file `Program1.java`. You may add methods to this file if you feel it necessary or useful. You may add additional source files if you so desire.

- `Driver.java` is the main driver program. Use command line arguments to choose between brute force and your efficient algorithms and to specify an input file. Use -ge for the efficient employee optimal algorithm, -gl for the effiecient location optimal algorithm and input file name for specified input (i.e. `java -classpath . Driver [-gl] [-ge] <filename>` on a linux machine). As a test, the 3-10-3.in input file should output the following for both a location and employee optimal efficient solution:

  - Employee 0 Location -1
  - Employee 1 Location 1
  - Employee 2 Location -1
  - Employee 3 Location -1
  - Employee 4 Location -1
  - Employee 5 Location -1
  - Employee 6 Location -1
  - Employee 7 Location 2
  - Employee 8 Location 0
  - Employee 9 Location -1

- When you run the driver, it will tell you if the results of your efficient algorithm pass the `isStableMatching()` function that *you coded* for this particular set of data. When we grade your program, however, we will use *our* implementation of `isStableMatching()` to verify the correctness of your solutions. We may test the correctness of the `isStableMatching()` separately.

- Make sure your program compiles on the LRC machines before you submit it.

- We will be checking programming style. A penalty of up to 10 points will be given for poor programming practices (e.g. do not name your variables foo1, foo2, int1, and int2).

- Before you submit, be sure to turn your report into a PDF and name your PDF file `eid_lastname_firstname.pdf`.

IMPORTANT: Submission Instructions

You should submit a single zip file titled `eid_lastname_firstname.zip` that contains all of your java files (no .class files) and pdf report `eid_lastname_firstname.pdf`. Do not put these files in a folder before you zip them (i.e. the files should be in the root of the ZIP archive). Your solution must be submitted via Canvas BY 11:59 am on Feb 14, 2020 to be on time. You will be allowed to submit the lab late until February 17th at 11:59PM for a 20 point penalty. No assignments will be accepted, for any reason, beyond that deadline.