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 OR Use a familiar programming language (C/C++)
- Compare the run time of an efficient algorithm against an inefficient algorithm
- Show an application of the stable matching problem

Problem Description

In this project, you will implement a variation of the stable marriage problem adapted from the textbook Chapter 1, Exercise 5, and write a small report. We have provided Java code skeletons that you will fill in with your own solution if you choose, or use another familiar programming language such as C/C++. Please read through this document and the documentation in the starter code thoroughly before beginning.

The Stable Matching Problem, as discussed in the text, assumes that all men and women have a fully ordered list of preferences. In this problem we will consider a version of the problem in which men and women can be indifferent between certain options. As before we have a set M of n men and a set W of n women. Assume each man and each woman ranks the members of the opposite gender, but now we allow ties in the ranking. For example (with $n = 4$), a woman could say that m_1 is ranked in first place; second place is a tie between m_2 and m_3 (she has no preference between them); and m_4 is in last place. We will say that w prefers m to m' if m is ranked higher than m' on her preference list (they are not tied).

We say that an assignment of men to women (a *matching*), S, is *weakly stable* unless there exists a man m and a woman w such that each of m and w strictly prefers the other to their partner in S. In other words, a match is *weakly stable* if there are people who are indifferent to whether they are with their current partners, or with each other, but not *weakly stable* if they prefer each other.

So we basically have the Stable Matching Problem as presented in class, except that preference lists may rank multiple people with the same priority, and with a slightly different definition of stability.

Part 1: Write a report [30 points]

Write a short report that includes the following information:

(a) Give an algorithm in pseudocode (as an outline) to find a stable assignment. *Hint: it should be very similar to the Gale-Shapley Algorithm*

(b) Give the runtime complexity of your algorithm in Big-O notation and explain why your proposal accurately captures the runtime of your algorithm.

(c) Consider a Brute Force Implementation of the algorithm where you find all combinations of possible matchings and verify whether they form a weakly stable marriage one by one. Give the runtime complexity of this brute force algorithm in Big O notation and explain why.

(d) In the following two sections you will implement code for a brute force solution and an efficient solution. In your report, use the provided data files to plot the number of couples (x-axis) against the time in ms it takes for your code to run (y-axis). There are four small data files and four large data files included in the input provided. The large data files may be too large for the brute force algorithm to finish running on your machine. If that is the case, do not worry about plotting the brute force results for the large data files. Your plot should therefore contain 8 points from your efficient algorithm and 4-8 points from the brute force algorithm. Please make sure the points from different algorithms are distinct so that you can easily compare the runtimes from the brute force algorithm and your efficient algorithm. Scale the plot so that the comparisons are easy to make (we recommend a logarithmic scaling). Also take note of the trend in run time as the number of hospitals increases.

Part 2: Implement a Brute Force Solution [30 points]

A brute force solution to this problem involves generating all possible permutations of students and hospitals, and checking whether each one is a stable matching, until a stable matching is found. For this part of the assignment, you are to implement a function that verifies whether or not a given matching is stable. We have provided most of the brute force solution already, including input, function skeletons, abstract classes, and a data structure. Your code will go inside a skeleton file called `Program1.java`. A file named `Matching.java` contains the data structure for a matching. See the instructions section for more information.

Inside `Program1.java` is a placeholder for a verify function called `isStableMatching()`. Implement this function to complete the Brute Force algorithm (the rest of the code for the brute force algorithm is already provided).

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. If you chose to code in C/C++, then ignore this comment.

Part 3: Implement an Efficient Algorithm [40 points]

We can do better than the above using an algorithm similar to the Gale-Shapley algorithm. Implement the efficient algorithm you devised in your report. Again, you are provided several files to work with. Implement the function `stableMarriageGaleShapley()` inside of `Program1.java`.

Of the files we have provided, please only modify `Problem1.java`, so that your solution remains compatible with the grading program. However, feel free to add any additional Java files (of your own authorship) as you see fit. If you chose to code in C/C++, then ignore this comment.

Instructions

- Download and import the code into your favorite development environment. Note for those using Java, we will be grading in Java 1.8. so make sure that your solution compiles with the standard Java 1.8 configuration (JDK 8). For most (if not all) students this should not be a problem. If you have doubts, email a TA or post your question on Piazza.
- If you do not know how to install Java or are having trouble choosing and running an IDE, post on Piazza for additional assistance or come speak to a TA.
- There are several `.java` files, but you only need to make modifications to `Program1.java`. You can modify `Driver.java` if it helps you to test or debug your program, as we will not grade the contents of `Driver.java`. **Do not modify the other files we have given you.** However, you may add additional source files to your solution if you so desire (just be sure to submit them all). 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. If not, there is probably a problem with your Java configuration.
- The main data structure for a matching is defined and documented in `Matching.java`. A Matching object includes:
  - **m**: The number of men.
  - **n**: The number of women.
  - **men_preference**: An ArrayList of ArrayLists containing each of the men's preferences for women. The women are listed in order from 0 to $n-1$ in these lists, and the numbers stored in the list represent the priority of how the man would prefer that woman. For example, if two women w_1 and w_2 are tied for first priority, and w_0 has second priority, then the preference list will read: $\{2, 1, 1\}$. The men are in order from 0 to $m-1$ in **men_preference**.
  - **women_preference**: An ArrayList of ArrayLists containing each of the women's preferences for men. The men are listed in order from 0 to $m-1$ in these lists. The format of the list is the same as above.
  - **woman_matching**: An ArrayList to hold the final matching. This ArrayList will hold the index of the man each woman is assigned to (in order of the women). 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 `setWomanMatching(<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 woman. The value at that index indicates to which man she is matched.
- You must implement the methods `isStableMaching()` and `stableMarriageGaleShapley()` 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 algorithm and to specify an input file. Use -b for brute force,

-g for the efficient algorithm, and input file name for specified input (i.e. `java -classpath . Driver [-g] [-b] <filename>` on a linux machine).

- 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 `verify()` to verify the correctness of your solutions.
- 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).
- If you are unsure how to do the plot in the report, we recommend the following resources. If you are on linux: `http://www.gnuplot.info/`. If you are using Windows: `http://www.online-tech-tips.com/ms-office-tips/excel-tutorial-how-to-make-a-simple-graph-or-chart-in-excel/`.
- We suggest using System.nanoTime() to calculate your runtime.
- Before you submit, be sure to turn your report into a PDF and name your PDF file `eid_lastname_firstname.pdf`.

What To Submit

You should submit a single zip file titled `eid_lastname_firstname.zip` that contains:

- all of your java files
- your 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 BEFORE 11:59pm on the day it is due.