COSC 2123/1285 Algorithms and Analysis
Semester 2, 2015

Assignment 2 Description

Due date: 11:59pm Wednesday, 21st October 2015 **Weight:** 15%

1 Background

AussieMatchMaker[1] is a dating company. They organise heterosexual dating events, and afterwards, their clients would provide their preferences for whom they would like to go on further dates. Up to now, they have done the matching manually, but clients are occasionally unhappy with their matches. In addition, some of their events can have hundreds of participants, and matching preferences for that many people is almost impossible by hand. Hence, they would like you to help them. The company is interested in the following problems:

1. Speed Dating: Given an event with n males and n females participants, each person has a preference ranking for the n members of the opposite sex. AussieMatchMaker would like to match each person to someone else of the opposite sex, if there are same number of males and females at a dating event. They would also like the matching to be stable. A *matching* is *stable* if for all matched couples, *both* individuals cannot be personally better off (matched to a more preferred partner) than their current match.

2. Bachelorette Dating: The AussieMatchMaker staff has been watching the Bachelorette, and thought they would try something similar. In Bachelorette dating, there are n males and m females (generally $n > m$). Each week a number of males date and try to win the approval of a female. In this form of dating, males and females still have a ranking preference to some of the opposite sex, but now each female can date/be matched to more than one male (every week, each female are allowed to choose the maximum number of guys they like to go out on dates).

In this project, you will help AussieMatchMaker by implementing algorithms to solve both problems.

2 Tasks

The project is broken up into a number of tasks to help you progressively complete the project.

Task A: Solve the Speed Dating Problem (8 marks)

The Speed Dating problem is an instance of the *stable marriage problem*. In the stable marriage problem, you have an equal number of males and females, who each have preferences to the members of the other sex. We seek a one-to-one stable matching between a male and female (see Background section of this assignment description, lecture notes or textbook for the definition of stable matching).

One popular algorithm to solve the stable marriage problem is the *Gale-Shapley* algorithm. It consists of a number of matching rounds, and can be described as follows (for the female proposing variant):

In the initial round:

[1] Fictitious company.

1. Each unmatched female proposes to the male she most prefers.
2. Each man replies “maybe” to the female he most prefers, and “no” to all other suitors.
3. Then the man is tentatively assigned to the female he most prefers, and that female is tentatively matched to him.

In subsequent rounds, until all females and males are matched:

1. Each unmatched female proposes to their most preferred male that they haven’t proposed to yet – this male could be tentatively matched already.
2. Each man replies “maybe” to the female he most prefers (this could be his existing tentatively partner or a new female) and rejects the rest.

When all females (and consequently males) have a tentative partner, this matching is stable and should be the final matching.

This algorithm is guaranteed to find a stable marriage configuration, and has $O(n^2)$ complexity. Your task is to implement the Gale-Shapley algorithm. You are free to use any existing data structures you know of, but the bipartite graph structure could be a good starting point.

Details

Your algorithm should be implemented in C, and the compiled program should be called “speed-DateMatch”. The program should take a command line argument that specifies the name of an input file, which will describe the preferences of males and females. More specifically:

```
speedDateMatch [preference file]
```

The format of the input preference file should be as follows. Let the number of males be n, and females be n. Then males are always assigned ids starting from 0 to $n-1$, while females also have ids from 0 to $n-1$. Sex and id uniquely identifies a client/participant. The format of the input file (it should be space separated CSV file) is as follows:

```
[number of males] [number of females]
0 [preferences of male id 0]
1 [preferences of male id 1]
...
[last male id] [preferences of male id last]
0 [preferences of female id 0]
1 [preferences of female id 1]
...
[last female id] [preferences of female id last]
```

To make it more clear, consider the following example. Let the number of males be 3, and females be 3. Let their preferences be specified in Figure 1.

Then the input file corresponding to this example would be:

```
3 3
0 0 1 2
1 1 0 2
2 1 2 0
0 0 2 1
1 2 1 0
2 2 0 1
```

Participant	**Preferences**
Male	
0	0 1 2
1	1 0 2
2	1 2 0
Female	
0	0 2 1
1	2 1 0
2	2 0 1

Table 1: Example, preferences of males and females.

Given an input preference file, your program "speedDateMatch" should then output to **stdout** the actual stable matching (there should only be one, if you implement the Gale-Sharley algorithm and have the females proposing). The actual matching should be in the following format:

```
[male id] [female id]
[male id] [female id]
...
```

Note that matching itself (list of matches) does not have to be in any particular order. Returning to our example, the output should be:

```
0 0
1 2
2 1
```

With a single space between each integer.

Note, please do not output anything else to stdout. If you need to print error messages or other output, print them to stderr. If you print something else to stdout, this may break automated testing and can result in you failing the tests.

Additional considerations

AussieMatchMaker wants to test different scenarios. They are as follows: Consider the following scenarios:

1. Sometimes in their events, they have different number of males and females. They would like to match as many couples as possible.
2. Sometimes their clients don't specify preferences for all the members of the other sex. They would like to match as many couples as possible.

In each of the scenarios, design and implement modifications to the Gale-Shapley algorithm. Consider how to come up with a new stable marriage – the matching may not be a complete matching (e.g., there are unmatched individuals), but the final matching should still be stable.

These scenarios will still use the same input file as described previously. But for the first scenario, there will be different number of males and females, hence some males/females will be unmatched. For the second scenario, some individuals may not have a preferences to some of the members of the opposite sex.

Task B: Solve the Bachelorette Dating Problem (5 marks)

The Bachelorette dating problem can be transformed to the college admission problem. Hence we describe it then present an algorithm to solve it.

The college admission problem consists of a number of students (e.g., trainee doctors) and a number of colleges (e.g., hospitals). Each college has a maximum number of places they can offer to students. Each student has preferences to each of the colleges, and each college have preferences to each student. The problem is to find a matching that is stable in terms of one party.

One famous algorithm for the college admission problem is the National Resident Matching Program (NRMP) algorithm. In the algorithm, a number of trainee doctors (applicants) have a list of preferences for different hospital programs. Each program also has a ranked list of candidates and may take more than one applicant (this is the main difference from the stable marriage problem).

The algorithm is as follows (for the version that favours applicants or is stable for applicants):

1. Each applicant is assigned their most preferred program. If the applicant cannot be matched to their first choice (due to program being full and they are lower ranked by the program than other applicants preferring that program), then they are matched to their second choice, until either the applicant has a tentative match or there are no more program choices on the applicants list and they become unmatched.
2. A tentative match occurs when
   (a) the program has the applicant on their preference list as well;
   (b) the program has an unfilled position or the program has no unfilled position but this applicant is more preferred by the program than one of the existing tentative matches to the program. The applicant less preferred by the program is removed from the program's positions and a tentative match is made to the more preferred applicant.
3. If an applicant is removed from a tentative match, the algorithm will attempt to re-match the applicant, starting from their most preferred program.
4. The algorithm loops until either all applicants are matched, all positions are filled or there are no free programs that unmatched applicants can be matched to (due to the free programs either not on the applicant's list or the applicant is not on the programs list).

Your task is to solve the Bachelorette dating problem by implementing the NRMP algorithm, assuming we favour males.

Details

Similar to Task A, your program implementing a solution for the Bachelorette dating problem should be named "bachDateMatch" should take as input an input file, specifying the preferences of clients. More specifically:

```
bachDateMatch [preference file]
```

Different from Task A, each female will specify the maximum number of dates they would like to go on for that week. The format is as follows:

```
[number of males] [number of females]
0 [preferences of male id 0]
1 [preferences of male id 1]
...
[last male id] [preferences of male id last]
0 [preferences of female id 0]
1 [preferences of female id 1]
...
[last female id] [preferences of female id last]
```

```
0 [maximum number of dates for female id 0]
1 [maximum number of dates for female id 1]
...
[last female id] [maximum number of dates for female id last]
```

To make it more clear, consider the following example. Let the number of males be n, and females be m. Let their preferences be illustrated in Figure 2:

Participant	**Preferences**	**Maximum number of dates**
Male		
0	0 1 2	-
1	1 0 2	-
2	1 2 0	-
Female		
0	0 2 1	2
1	2 1 0	1
2	2 0 1	3

Table 2: Example, preferences of males and females, with the maximum number of dates that each female can attend in that week.

Then the input file corresponding to this example would be:

```
3 3
0 0 1 2
1 1 0 2
2 1 2 0
0 0 2 1
1 2 1 0
2 2 0 1
0 2
1 1
2 3
```

Your program "bachDateMatch" should then output to stdout the actual matching.

```
[male id] [female id]
[male id] [female id]
...
```

Note that matching itself (list of matches) does not have to be in any particular order. Returning to our example, there is a stable matching so the output could be:

```
0 0
1 0
2 1
```

With a single space between each integer.

Note, please do not output anything else to stdout. If you need to print error messages or other output, print them to stderr. If you print something else to stdout, this may break automated testing and can result in you failing the tests.

3 Submission

The final submission will consist:

- Your C source code (source .c and header .h files) and Makefile of your implementations (we do not need the object files, executables nor core dumps). Your makefile should have two targets, one called "speedDateMatch", the other "bachDateMatch". When "make speedDateMatch", this should create an executable "speedDateMatch" and should implement your algorithm for Task A. When "make bachDateMatch", this should create an executable "bachDateMatch" and should implement your algorithm for Task B.

  When submitting your assignment, place all your code and Makefiles into a folder called Assign2-<your student number>, where "student number" is your student number, e.g., if my student number is s1234, then the folder be called "Assign2-s1234". Then zip this folder up and call it Assign2-<your student number>, such that when we unzip your zip file, a folder called Assign2-<your student number> will be created, and inside will be your code and Makefile. Ensure your code complies with either ANSI (C90) or C99 standards, and compiled using gcc -Wall -pedantic.

  As an example, if your student number is s1234, and we got your submission as Assign2-s1234.zip, then on the command line we should be able to compile your program as follows:

  ```
  > unzip Assign2-s1234.zip
  > cd Assign2-s1234
  > make speedDateMatch
  > make bachDateMatch
  > ls -1
   ...
  speedDateMatch
  bachDateMatch
   ...
  ```

  Please conform to this submission format. If it is not in this format, the automated scripts are likely to break and unlike assignment 1, we will not spend time on finding how you structured your code and try to manually compile them.

Note: submission of the code will be done via WebLearn. We will provide details closer to the submission deadline.

4 Assessment

The project will be marked out of 15. There are **2 marks** allocated for *code documentation and design*. Note that there is a hurdle requirement on your combined continuous assessment mark for the course, of which Project 2 will contribute 15 marks. Late submissions will incur a deduction of 3 marks per day, and no submissions will be accepted 5 days beyond the due date.

5 Plagiarism Policy

University Policy on Academic Honesty and Plagiarism: You are reminded that all submitted project work in this subject is to be your own individual work. It is not a group project. Multiple automated similarity checking software will be used to compare submissions. It is University policy that cheating by students in any form is not permitted, and that work submitted for assessment purposes must be the independent work of the student(s) concerned. Plagiarism of any form will result in zero marks being given for this assessment, and can result in disciplinary action.

For more details, please see the policy at `http://rmit.info/browse;ID=sg4yfqzod48g1`.

6 Getting Help

There are multiple venues to get help. There are weekly consultation hours (see Blackboard for time and location details). In addition, you are encouraged to discuss any issues you have with your Tutor or Lab Demonstrator. We will also be posting common questions on Blackboard and we encourage you to check and participate in the discussion forum on Blackboard. Although we encourage participation in the forums, please refrain from posting solutions.