Université d'Ottawa
Faculté de génie

École d'ingénierie et de technologie de l'information

uOttawa

L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Information Technology and Engineering

Programming Assignment 1 (15%) CSI2110/CSI2510
PART 1 (0.70*60 marks=42marks=10.5%)
Due : October 24, 11:59PM

Politique de retard: ***1min-24hres de retard -30% pénalité; devoirs non acceptés après 24hres.***

Late assignment policy : ***1min-24hs late are accepted with 30%off; no assignments accepted after 24hs late.***

The Stable Matching problem

Problem Description

This assignment asks you to implement a solution to the ***stable matching problem***. In our case, we will consider the problem of matching coop employers to coop students. In particular, we consider the problem of `n` employers and `n` students where each employer will hire a single student. Each employer produces a ranking of students according to their preference to hire for the coop term and each student rank all employers according to their preference to work for. Based on these preferences your mission is to write a program that will match each employer with one student such that all players will be as satisfied as possible with their match (in the sense that there will be no incentive for a pair of employer-student that are not matched to each other, to both agree to be matched).

You will be given a list of employers. For example

```
0. Thales
1. Canada Post
2. Cisco
```

And a list of students. For example:

```
0. Olivia
1. Jackson
2. Sophia
```

The two lists will always be of the same size.

Students and employers will enter their ranking. For example:

- Employer preferences :
  ```
  0. Thales:         0.Olivia, 1.Jackson, 2.Sophia
  1. Canada Post:   2. Sophia, 1.Jackson, 0.Olivia
  2. Cisco: 0.Olivia, 2.Sophia, 1.Jackson
  ```

- Student preferences:
  ```
  0. Olivia:   0.Thales, 1.Canada Post, 2.Cisco
  1. Jackson:  0.Thales, 1.Canada Post, 2.Cisco
  2. Sophia:   2.Cisco, 0.Thales, 1.Canada Post
  ```

These ranking will be given to you in a text file formatted accordingly.

A stable matching solution for this problem is as follows:

```
Thales - Olivia
Canada Post - Jackson
Cisco - Sophia
```

Why is this solution stable? We have to look at the definition of a stable match. A stable match is defined such that neither party to the match has a preferred party that also prefers them over their current match. E,g., Canada Post would prefer to hire Olivia over Jackson but Olivia is currently matched with Thales which she prefers over Canada Post. While Jackson would prefer to work for Thales over Canada Post but Thales is matched with Olivia which Thales prefers over Jackson. As a result, while neither Canada Post nor Jackson got their first choice, the match is stable as neither of them have a way to improve their current match.

As a result, the solution provided is a stable matching and is also perfect. A perfect match is actually a simpler criterion, just requiring each employer being matched to a student and each student being matched to an employer.

In summary, in this assignment you will need to find a perfect and stable matching given preference tables by coop employers and by students. There will always be the same number of employers and students and every employer will only hire one student.

Algorithm to be used

The stable matching can be found with an iterative algorithm, the Gale-Shapley algorithm. The pseudocode implementing this algorithm is given below. The input will be given as a simple text file. It will contain the list of employers, the list of students and an `nxn` matrix of ranking pairs.

Gale-Shapley algorithm

Input:

- A list of `n` employers, indexed `0` to `n-1`
- A list of `n` students, indexed `0` to `n-1`
- An `nxn` matrix in which each entry is a pair (*employer_ranking*, *student_ranking*); (each ranking being in the interval [1,n]). One row corresponds to one employer. One column corresponds to one student.

Initialization:

- Create `Sue`, a stack of unmatched employers
- Push all employers in this stack starting from employer `0`
- Create two arrays of size `n`, `students` and `employers`, used to represent the current matches; (if student `s` is matched with employer `e` then `students[s]=e` and `employers[e]=s`; value `-1` is used to designate an unmatched student or employer)
- Initialize all entries in these arrays to `-1`
- Create 2D array `A` of size `nxn` with `A[s][e]` being the ranking score given by student `s` to company `e`
- Initialize each entry with the *student rankings*.
- Create `n` priority queues with `PQ[e]` being the queue of employer `e`
- `For each student s`
    - `For each employer e`
        - `PQ[e].insert(`*employer_ranking*`,s)`

Procedure:

- while (!Sue.empty())
  - o e= Sue.pop() // e is looking for a student
  - o s= PQ[e].removeMin() O(n) // most preferred student of e
  - o e'= students[s]
  - o if (students[s] == -1) // student is unmatched
    - ▪ students[s]= e
    - ▪ employers[e]= s // match (e,s)
  - o else if (A[s][e] < A[s][e']) // s prefers e to employer e'
    - ▪ students[s]= e
    - ▪ employers[e]= s // Replace the match
    - ▪ employers[e']= -1 // now unmatched
    - ▪ Sue.push(e')
  - o else s rejects offer from e
    - ▪ Sue.push(e)
- return the set of stable matches

Refer to the annex for an example of this algorithm in action.

Input file format

A text file will be given as input. The first number will be the number of students and employers, followed by the list of employers, the list of students and a matrix of ranking pairs (employer ranking, student ranking). Employers correspond to the rows of this matrix and students correspond to columns. For the simple example given on the first page, this input file will be as follows:

```
3
Thales
Canada Post
Cisco
Olivia
Jackson
Sophia
1,1 2,1 3,2
3,2 2,2 1,3
1,3 3,3 2,1
```

Output file format

Your program must simply produces as output a textfile. If the input file is called ABC.txt then the output file must be called `matches_ABC.txt` that contains the list of match pairs `employers - students` keeping the same order for the company names as listed in the input file.

```
Match 0: Thales - Olivia
Match 1: Canada Post - Jackson
Match 2: Cisco - Sophia
```

You must follow this exact format in which matches are listed from `0` to `n-1` (match `i` being the match for company `i`). Company and student names are separated by a dash symbol (`-`). If you do not follow this format, your solution will not be evaluated.

Requirements

- You must create a program following the Gale-Shapley algorithm exactly as described above. You have to use the data structures and the procedure as described. You cannot propose your own variant of the algorithm.
- You must create a class called Gale-Shapley with the data structure `Sue, PQ, A, students, employers` as described. For the implementation of these ADTs, you are free to use the standard Java classes, the implementations given in the book or in the class notes or using your own implementations. But the stack used must have `pop` and `push` methods each running in O(1) and the priority queue must have `insert` and a `removeMin` methods each running in O(log n).
- Your class must have an `initialize(filename)` method that reads the input file and performs all initialization steps as described above.
- Your class must have an `execute()` method that perform the Gale-Shapley algorithm as described above.
- Your class must have a `save(filename)` method that saves the results in text file as described.
- The main method of this class simply asks for a filename, then calls the methods `initialize`, `execute` and `save`.
- All your Java files must have a header that includes your name and student number.
- Your Java files must be appropriately commented. Include all your Java files in a zip file called projectCSI2110_XXX.zip where XXX is your student number. Include all files required for your program to compile.
- Include also in your zip file the output files showing the solutions for all input files provided.

Marking Scheme

Correctness of solutions provided:	15%
Quality of programming:	15%
Initialize and save method:	10%
Execute method:	20%
Abstract data types used:	10%
Supplementary questions (Part II):	30% (to be published later: 0.30*60marks=18marks=4.5%)

Annex: algorithm walkthrough

This is a step-by-step analysis of the algorithm for the simple case given on pages 1-2.

```
Sue= [ 0  1  2
Students= {-1, -1, -1}
Employers= {-1, -1, -1}
A= {1,  2,  3}
   {1,  2,  3}
   {2,  3,  1}
PQ[0]= {(1,0),(2,1),(3,2)}
PQ[1]= {(3,0),(2,1),(1,2)}
PQ[2]= {(1,0),(3,1),(2,2)}
```

Step 1:

```
2 <- Sue.pop()
Sue= [ 0  1
0 <- PQ[2].removeMin()
PQ[2]= {(3,1),(2,2)}
-1 <- e'=Students[0]
Students[0]= 2
Employers[2]= 0
Match: Cisco-Olivia
```

Step 2:

```
1 <- Sue.pop()
Sue= [ 0
2 <- PQ[1].removeMin()
PQ[1]= {(3,0),(2,1)}
-1 <- e'=Students[2]
Students[2]= 1
Employers[1]= 2
Match: CanadaPost-Sophia
```

Step 3:

```
0 <- Sue.pop()
```

```
Sue= [
0 <- PQ[0].removeMin()
PQ[0]= {(2,1),(3,2)}
2 <- e'=Students[0]
(1<-A[0][0]) < (3<-A[0][2])
Students[0]= 0
Employers[0]= 0
Employers[2]= -1
Sue= [ 2
Match: Thales-Olivia
```

Step 4:

```
2 <- Sue.pop()
Sue= [
2 <- PQ[2].removeMin()
PQ[2]= {(3,1)}
1 <- e'=Students[2]
(1<-A[2][2]) < (3<-A[2][1])
Students[2]= 2
Employers[2]= 2
Employers[1]= -1
Sue= [ 1
Match: Cisco-Sophia
```

Step 5:

```
1 <- Sue.pop()
Sue= [
1 <- PQ[1].removeMin()
PQ[1]= {(3,0)}
-1 <- e'=Students[1]
Students[1]= 1
Employers[1]= 1
Match: CanadaPost-Jackson
```

Sue is empty!