Homework 1

Due by **12:30pm, Friday, September 16, 2016**.

Make sure you follow all the homework policies (http://www-student.cse.buffalo.edu/~atri/cse331/fall16/policies/hw-policy.html).

All submissions should be done via Autolab (http://www-student.cse.buffalo.edu/~atri/cse331/fall16/autolab.html).

1. Just in there is any confusion: this is a totally made up scenario: I'm not claiming the ability to predict the future.

Some Questions on Stable Matching

Sample Problem

The Problem

Decide whether the following statement is true or false:

In every Stable Marriage problem instance where a man m and woman w have each other as their least preferred partner, the following is true. There is no stable matching for the instance where (m, w) are matched.

If you state true then you will have to formally argue why the statement is correct. If you state false, then you have to give a counter-example.

Click here for the Solution

More Practice Questions

The Problem

Exercises 1 and 2 in Chapter 1.

Submission

You will **NOT** submit this question. This is for you to get into thinking more about the stable matching problem.

Question 1 (Programming Assignment) [40 points]

</> Note

This assignment can be solved in either Java, Python or C++ (you should pick the language you are most comfortable with). Please make sure to look at the supporting documentation and files for the language of your choosing.

Given an input for the stable matching problem, output **all** stable matchings for that instance.

Input

The input is an instance of the stable marriage problem in a text file of the following format:

```
n                                   <- Number of men and women (each)
m1^1  m2^1  m3^1 ...  mn^1           <- Preference of the 1st woman (most preferred first)
m1^2  m2^2  m3^2 ...  mn^2           <- Preference of the 2nd woman (most preferred first)
m1^3  m2^3  m3^3 ...  mn^3           <- Preference of the 3rd woman (most preferred first)
m1^4  m2^4  m3^4 ...  mn^4           <- Preference of the 4th woman (most preferred first)
.
.
.
m1^n  m2^n  m3^n ...  mn^n           <- Preference of the nth woman (most preferred first)
w1^1  w2^1  w3^1 ...  wn^1           <- Preference of the 1st man (most preferred first)
w1^2  w2^2  w3^2 ...  wn^2           <- Preference of the 2nd man (most preferred first)
w1^3  w2^3  w3^3 ...  wn^3           <- Preference of the 3rd man (most preferred first)
w1^4  w2^4  w3^4 ...  wn^4           <- Preference of the 4th man (most preferred first)
.
.
.
w1^n  w2^n  w3^n ...  wn^n           <- Preference of the nth man (most preferred first)
```

For example

```
4                                   <- Number of men and women (each)
3   4   2   1                       <- Preference of the 1st woman (most preferred first)
2   4   3   1                       <- Preference of the 2nd woman (most preferred first)
2   3   4   1                       <- Preference of the 3rd woman (most preferred first)
4   1   2   3                       <- Preference of the 4th woman (most preferred first)
1   2   3   4                       <- Preference of the 1st man (most preferred first)
3   1   4   2                       <- Preference of the 2nd man (most preferred first)
2   1   4   3                       <- Preference of the 3rd man (most preferred first)
4   1   2   3                       <- Preference of the 4th man (most preferred first)
```

Output

The output is the set of **all** stable matchings for the input in a text file of the following format:

```
x                                                   <- Number of stable matchings found
[(m1^1,1),(m2^1,2),(m3^1,3),(m4^1,4),...,(mn^1,n)]     <- Pairing of the form (m,w)
[(m1^2,1),(m2^2,2),(m3^2,3),(m4^2,4),...,(mn^2,n)]     <- Each stable matching is sorted by women (1...n)
[(m1^3,1),(m2^3,2),(m3^3,3),(m4^3,4),...,(mn^3,n)]
.
.
.
[(m1^x,1),(m2^x,2),(m3^x,3),(m4^x,4),...,(mn^x,n)]     <- xth stable matching
```

Please note that each stable matching outputted is sorted by women first. For example:

```
3                                   <- Number of stable matchings found
[(3,1),(1,2),(2,3),(4,4)]           <- Pairing of the form (m,w)
[(4,1),(2,2),(1,3),(3,4)]
[(4,1),(3,2),(1,3),(2,4)]
```

Please note that the above output is an example is not the correct solution to the input example given above. For correct pairs of input and output, please look at any of the code skeletons.

Hint

The best possible algorithm for this problems that we are aware of runs in time $O(n^2 \cdot n!)$. This is not terribly efficient but an exponential runtime for this problem is unavoidable-- see Question 3. For this reason we will be testing your implementations with testcases on $n \leq 10$.

It *might* be useful to implement Heap's algorithm (../../../support/proofs/index.html) as one of the functions that your algorithm implementation uses.

! Note

Both the input and output parsers in each of the three languages are already written for you.
Note that you have to work with the input data structures provided (which will come pre-loaded with the data from an input file). Also note that you do not have to sort your output: we'll take care of that.

! Addition is the only change you should make

Irrespective of what language you use, you will have to submit just one file. That file will come pre-populated with some stuff in it. You should **not** change any of those things because if you do you might break what the grader expects and end up with a zero on the question. You should of course add stuff to it (including helper functions and data structures as you see fit).

Please Note

We tried to polish the helper material as much as we could but they of course can be improved. If you have any suggestions for improvement and/or need some clarification, please ask on piazza!

We do not have the full functionality of Autolab yet, so we will have separate submission links for each language. From HW 2 onwards, we are hoping to have a single submission link, where you can choose your language from a drop-down menu.

As a corollary of the above, for Questions 2 and 3 you will have to explicitly write down your sources and collaborators in your submissions. From HW 2 onwards, we are hoping that you submit this via a form when you upload your PDF.

Java | Python | C++

Download Java Skeleton Code (HW1Java.zip)

Directory Structure

```
├── Driver.java
├── HW1_Student_Solution.java
├── HW1_Utils.java
├── Marriage.java
├── testcases/
│   ├── input1.txt
│   ├── input2.txt
│   ├── input5.txt
│   ├── output1.txt
│   ├── output2.txt
│   └── output5.txt
```

You are given four coding files: `Driver.java`, `HW1_Utils.java`, `Marriage.java` and `HW1_Student_Solution.java`. `Driver.java` takes the input file, parses it and creates an instance of the class `HW1_Utils` and calls the `outputStableMatchings()` method on it. It then prints all the computed perfect matchings (which, if your code is correct, will all be stable matchings). You only need to update the `HW1_Student_Solution.java` file. You may write your own helper methods and data structures in it.

The testcases folder has 3 input files and their corresponding output files for your reference. We will use these three input files (and seven others) in our autograding.

Method you need to write:

```
5.
/**
 * This method must be filled in by you. You may add other methods and subclasses
 * but they must remain within the HW1_Student_Solution class.
 * @return Your set of stable matches. Order does not matter.
 */
public ArrayList<ArrayList<Marriage>> outputStableMatchings() {

  return _stableMatchings;
}
```

The `HW1_Student_Solution` class has 4 instance variables.

- `_numberOfMenAndWomen` which is of type int and stores n.
- `_men` which is of type `HashMap<Integer, ArrayList<Integer>>` and stores the preference lists of men. Please note that the front of the `ArrayList<Integer>` (index 0) denotes the most preferred partner.
- `_women` which is of type `HashMap<Integer, ArrayList<Integer>>` and stores the preference lists of women. Please note that the front of the `ArrayList<Integer>` (index 0) denotes the most preferred partner.
- `_stableMatchings` which is of type `ArrayList<ArrayList<Marriage>>` and stores the set of stable matchings you find for the given instance.

The Other files

`Marriage.java` defines the `Marriage` class. This should be fairly intuitive. Below is the entire content of the class:

```
                                    public class Marriage {
                                        public Integer _man;
                                        public Integer _woman;
                                        Marriage(Integer man, Integer woman){
 5.                                         _man = man;
                                            _woman = woman;
                                        }

                                        /**
10.                                      * Used to compare if two marriages match.
                                         * @param compare The other Marriage that this is being compared to.
                                         * @return true if they share the same man & woman
                                         */
                                         public boolean equals(Marriage compare){
15.                                        return (_man == compare._man) && (_woman == compare._woman);
                                         }
                                         public String toString(){
                                           return "(" + _man + ", " + _woman + ")";
                                         }
20.                                 }
```

The file `HW1_Utils.class` handles some of the background stuff: e.g. `readFile` reads in the file passed as a command line argument to `Driver.java` and populates the preference lists within the `HW1_Utils` class. It also provides `sortMatchingList` to sort a list of perfect matchings and `compare` to check if two lists of perfect matchings are the same. (You should not need to use the latter but it is provided for your convenience.)

Compiling and executing from command line:

Assuming you're in the same directory level as `Driver.java`. Run `javac Driver.java` to compile.

To execute your code on `input1.txt`, run `java Driver testcases/input1.txt`.

Submission

You only need to submit `HW1_Student_Solution.java` to Autolab.

Grading Guidelines

We will follow the usual grading guidelines for programming questions (../../policies/hw-policy.html#grading).

Question 2 (Stable Movie Schedules) [45 points]

The Problem

We are some years into the future and movie theaters as we know it now have disappeared. [1] In the future, all movies will be released online on a platform owned by `MegaInternetVideoContentProvide`. Every movie studio (yes, these still exist in this future) have to schedule the release dates of their movies with `MegaInternetVideoContentProvide`. More than one movie can be released on one day.

In particular, for now let us consider the case of two studios A and B that have n movies in their roster that they want to schedule on the same n release dates on `MegaInternetVideoContentProvide`.

Since `MegaInternetVideoContentProvide` had access to the movie preferences of every human being on the planet, given a movie m it can accurately predict a *success number* for m. You can assume that the success number is an integer and a larger success number means the larger chance of the movie being watched on `MegaInternetVideoContentProvide` (and hence make more money). We say for a given release date a studio *demolishes* the other studio if its released movie on that release date has a higher success number than any other movie released on the same day.

> #### What about ties?
> If on a given date both studios release movies with the *same* success number, then you should assume that studio A demolishes studio B.

At the beginning of each fall, studio A reveals a schedule S and studio B reveals a schedule T. (From now on you should assume that these are the only two studios that exist in the future.) A schedule for a studio assigns each of its n movies to the n release dates. Thus, given the schedules S and T, each release date has exactly two movies being released on that date: one from studio A and another from studio B. The goal of the studios in designing the schedules S and T is to demolish the other studio on as many release dates as possible.

In this problem, you will explore the notion of stability in this context. In particular, a given pair of schedule (S, T) is said to be stable if no studio can unilaterally change its own schedule and end up demolishing the other studio on more release dates. More precisely, the pair of schedule (S, T) is stable if for any alternate set of schedules S' for studio A, studio A demolishes studio B on no more release dates with the pair of

schedules (S', T) than with the pair (S, T). Further, for any other schedule T' for studio B, studio B demolishes studio A on no more release dates with the pair of schedules (S, T') than with the pair (S, T).

Here is an example to illustrate the above. Consider the case of $n = 3$ and A has movie/success number pairs of $(m_1, 100), (m_2, 101), (m_3, 102)$ while B has pairs $(m_4, 1), (m_5, 2), (m_6, 3)$, where each m_i is a movie and the corresponding number is its success number. In this case *every* pair of schedule (S, T) is stable (since A wins on all the release dates irrespective of the schedule).

The corresponding question to one we asked in class is the following: For **every** set of n movies and their corresponding success numbers, does there always **exists** a stable pair of schedules? Resolve this question in one of the following two ways:

- Give an algorithm that for any set of movies and their corresponding success numbers, computes a stable pair of schedules; or
- Give an example of movies and their corresponding success numbers, for which there does not exist any stable pair of schedules.

Note

It is helpful for this problem, to formally write down in first order logic, what the problem is asking for. To help you guys out, here is an overview of what you need to do:

- If you are providing an algorithm then for **every** input, your algorithm should output a pair of schedules (S, T) that **is stable**. You will also have to prove that your algorithm outputs a pair of stable schedules on every input.
- if you are providing a counter-example, then you have to present only **one** input and then prove that **every** possible pair of schedules (S, T) is ***NOT* stable**.

Submission

You need to submit **one PDF** file to Autolab. We recommend that you typeset your solution but we will accept scans of handwritten solution-- you have to make sure that the scan is legible.

Grading Guidelines

We will follow the usual grading guidelines for programming questions (../../policies/hw-policy.html#grading). Here is a high level grading rubric specific to this problem:

1. `Proof idea : 23` points for a proof/counterexample idea explaining the insight behind why you think the property holds or not.
2. `Proof details : 22` points for a detailed proof or counterexample.

! Note

If you do not have separated out proof idea and proof details, you will get a zero(0) irrespective of the technical correctness of your solution..

! Note

Your must explicitly list your sources and collaborators on your submission. Note that you can only used one of the five allowed sources (../../policies/hw-policy.html). If you have used a source that is not allowed, please do **not** submit your homework. **If you do not have clearly marked sources or collaborators, you will get a zero(0) irrespective of the technical correctness of your solution.** If you did not consult any source or did not collaborate with anyone just say `None` .

Questions 3 (Many Stable Matchings) [15 points]

The Problem

For every $n \geq 2$ that is a multiple of 2, show that there is a stable matching instance on n men and n women such that the instance has at least $2^{n/2}$ distinct stable matchings.

To get **full credit**, you should be able to present an instance for *every* $n \geq 2$ that is a multiple of 2.

Hint

First try and construct such an instance for $n = 2$. Try and extend the instance to every n that is a multiple of 2.

Note

If you show a family of instances with (many) more stable matchings than $2^{n/2}$ that is of course fine. In fact, if you are looking for a puzzle, try and prove as large a lower bound on the number of stable matching as you can. If you get something better than say 3^n come talk to me. **Warning**: Showing such a bound will solve an open problem, so be warned that doing so might not be easy.

Submission

You need to submit **one PDF** file to Autolab. We recommend that you typeset your solution but we will accept scans of handwritten solution-- you have to make sure that the scan is legible.

Grading Guidelines

We will follow the usual grading guidelines for programming questions (../../policies/hw-policy.html#grading). Here is a high level grading rubric specific to this problem:

1. `Proof idea : 8` points for outlining the proof idea.
2. `Proof details : 7` points for the proof of the general case (any $n \geq 2$).
3. `Note` : If your solution only presents example(s) for some *specific* $n \geq 2$, then you get at most 2 points for the entire question.

! Note

If you do not have separated out proof idea and proof details, you will get a zero(0) irrespective of the technical correctness of your solution..

! Note

Your must explicitly list your sources and collaborators on your submission. Note that you can only used one of the five allowed sources (../../policies/hw-policy.html). If you have used a source that is not allowed, please do **not** submit your homework. **If you do not have clearly marked sources or collaborators, you will get a zero(0) irrespective of the technical correctness of your solution.** If you did not consult any source or did not collaborate with anyone just say `None` .