# CS 330 – Fall 2021
# Homework 3, Problem 2: Stable Matching
# Due September 22, 2021

In this exercise you will work with the Gale-Shapley algorithm to solve the Stable Matching problem found in Section 1.1 of *Algorithm Design*.

There are three parts, for which we provide detailed instructions below.

1. Complete an implementation of Gale-Shapley so that it runs in $O(n^2)$ time.
2. Implement an algorithm that takes as input the preference lists for $n$ hospitals and students as well as a candidate matching, and checks if the matching is stable.
3. Implement an algorithm that takes as input the preference lists for $n$ hospitals and students and decides if, for these preferences, there is only one stable matching (as opposed to multiple possible stable matchings).

You will write your code in Python 3.x and submit via Gradescope. The starter code, available on Piazza, is also in Python 3.x. (Specifically, we are testing with Python 3.8.5, but code compatible with 3.6 or higher should work).

**IMPORTANT: Generating Your Own Tests** For each part, we provide several inputs and their corresponding outputs. You can use these to test your code and to make sure you understand what you are supposed to produce. However, *do not rely on only our test inputs to test your code!* We give you a few test examples to get you started, but you are responsible for making sure that your program is really correct. You can and should write out an argument for why your program is correct. You should also devise and run your own tests. For longer programs, it's a good idea to break your program into several parts, and then test each part separately.

**Submission Format** On Gradescope submit a file `stable_matching.py` (name must match exactly) that answers the thee questions of this assignment. A starter code version of `stable_matching.py` is given.

## 2.1 Completing the Gale-Shapley Implementation

We provide starter code with a slow implementation of Gale-Shapley. It runs in time $\Theta(n^3)$. For Part 1, you should modify the implementation so it runs in time $O(n^2)$.

**Input format** You should test your file using the command:

```
python stable_matching.py pref_file_1 pref_file_2 out_name Q1
```

- The files `pref_file_1` and `pref_file_2` contain the preference lists of the two parties (which we will call "hospitals" and "students"), respectively. In the starter code for Q1, the first preference list will be the proposing party (hospitals). We index both groups with 0 to $N-1$. `pref_file_1` contains $N+1$ rows. The first row is the integer $N$, the number of students. The rows after this give the preference list for each hospital. Take row $i+1$, which gives the hospital $i$'s preference list for all the students. The $j$th number in this row is $i$'s $j$th most preferred student.

- `pref_file_2` is structured the same way: the first line contains the integer $N$ with the remaining lines specifying the preference lists for each student.
- `out_name` specifies the file name in which output will be printed.
- `Q1` specifies that we are testing question 1.

**Output format** The output will be stored by the path and name given from `out_name`. From the starter code for question 1, it will be a text file with each line specifying one matched pair. Each pair is specified by the index of a hospital followed by the index of a student, separated by a comma.

**Instructions** The implementation we provide has one step in the main loop that runs in linear time (it's identified in the comments). Fix that step by preprocessing the preferences lists for the side receiving proposals ("students") as described in class. If your new code runs in time $O(n^2)$, you should see a speed-up of roughly a factor of $n$ when running on inputs of length $n$, especially for preference lists that causes many proposals. In particular, the starter code takes a few minutes to run with $n = 3,000$ students and hospitals (for the input where all hospitals have the same preferences) on the machine we used for tests. The improved version runs in under 10 seconds.

**Test Input/Output Files** The starter code contains several test inputs along with valid outputs (i.e., stable matchings). These are the outputs that the slow implementation will generate. They have sizes $N = 10, 100, 1000, 500, 3000$. Your fast implementation should generate the same outputs (unless you change more than just the parts we suggest).

As mentioned above, the test input/output pairs we give you are just to get you started. You should generate and run your own tests!

## 2.2 Checking if a Matching is Stable

In this part, you will write a program that takes two kinds of inputs: the preference lists and a candidate matching. Your program should print `1` if the matching is perfect (meaning every hospital is matched to exactly one student and vice versa) and stable, and it should output `0` otherwise.

**Input format** For part 2, the program takes three input files: two preference files (formatted as in question 1) and a candidate matching (formatted the same way as the output from Part 1).

We will run your program using the command:

```
python stable_matching.py pref_file_1 pref_file_2 match_file Q2
```

The starter code has a placeholder for your code for Part 2.

**Output format** Your program should print either `0` or `1` for unstable and stable, respectively. [1]

**Instructions** Your program should run in time $O(n^2)$ for inputs of with $n$ hospitals and students. It should be roughly similar to the running time of your improved Gale-Shapley implementation from Part 1.

[1] As you are testing your program, you might want it to print other information to help you find mistakes. But the final submission file must only print 1 number (`0` or `1`).

**Test Input/Output Files** We provided test preference lists and matchings of sizes $N = 10, 100, 1000, 500, 3000$. A plain text file is included with the test inputs to indicate the correct outputs.

## 2.3 Checking if a Unique Stable Matching Exists

In this part, you will write a program that takes in preference lists and decide whether a unique stable matching exists for the two parties. print `1` if a unique stable matching exists, and it should output `0` otherwise.

**Input format** The input format is the same as for question 1. We will test your code using the command:

```
python stable_matching.py pref_file_1 pref_file_2 ignored Q3
```

(The third argument has to be there for compatibility with the first two questions, but it will be ignored.) The starter code has a placeholder for your code for Part 3.

Your program should run in time $O(n^2)$ for inputs of with $n$ hospitals and students.

**Output format** Your program should print either `0` (for not unique) or `1` (for unique), respectively.

**Test Input/Output Files** We provided test preference lists of sizes $N = 10, 100, 1000, 500, 3000$. A plain text file is included with the test inputs to indicate the correct outputs.