CSC 440 Assignment 1: Perfect Marriage in King Arthur's court, and Pandemic

Out: Thursday, January 26th

Due: Monday, February 6th, by 11:59PM

Introduction

The purpose of this ***pair*** assignment is threefold:

1. To shake off the cobwebs if your Python is a little rusty
2. To implement a graph algorithm based on pseudocode for the algorithm, using ***appropriate choices of data structures given the access patterns***
3. To begin learning how ***invariants*** can help you reason about the correctness of an algorithm.

Part 1: Perfect Marriage in King Arthur's Court

King Arthur issues a decree: all the knights and ladies in Camelot must marry[1]. As it happens, there are an equal number, so he insists this should be possible. Arthur orders Merlin, the magician, to come up with an *algorithm* to match the knights and ladies up so that no marriage is *unstable*. Merlin comes up with an algorithm:

- Every knight ranks the ladies in order of preference.
- Similarly, every lady ranks the knights in order of preference.
- Each knight proposes to the lady he prefers most.
- Then, each lady either considers all the proposals she has received, and replies "maybe" to the knight she likes best (of the ones that proposed to her) and "no" to all the rest. Thus, she is now *engaged* to that knight.
- As long as there are unengaged knights, each unengaged knight proposes to the most-preferred lady to whom he has not yet proposed, regardless of whether or not she is engaged.

[1] Why no marriage equality? Well, this is a fictional story, and it turns out the Gale-Shapley algorithm requires two different "classes". If you prefer, you can refer to one class as medical residencies and the other as medical doctors.

- Each lady reviews the new proposals, and either replies "maybe" if she is not yet engaged or if she prefers this knight to the one she was previously engaged to (if she rejects her previous knight, he becomes unengaged)

Merlin proves to King Arthur that as long as there are the same number of knights and ladies, this algorithm will result in a perfect, stable marriage: everyone will be married, and there will not exist some knight and some lady who would both rather be with each other than with their current partner.

We discussed this algorithm, called *Gale-Shapley*, and proved that its asymptotic complexity is $O(n^2)$, as well as that the algorithm provides a *stable marriage*. In most future assignments, you will have to spend some time proving complexity bounds, but for this one, you will not. Instead, you will:

- Implement Gale-Shapley in Python.
- Document the invariants in your code
- Benchmark your implementation on data sets of different sizes, which we provide
- Plot the running time versus the input size, and explain whether or not the results meet your expectations.

You will need to upload your code (python file) as well as the PDF containing your Pandemic solution, and benchmarks and explanation of your program's running time (either separate or combined PDFs are fine).

Data File Format and Program Arguments

It is essential to understand that you are writing software whose output will be consumed by other software (namely, the autograder). Thus, your output must conform *exactly* to the specification given here, just as we promise the input files we give you will conform exactly to the input specification we give.

Your program **must** be called `marriage.py` and **must** take a single command-line argument, which specifies an input file.

Failure to adhere to these specifications will result in a **zero** grade for Functional Correctness.

We provide a zip archive on EdStem, `marriage_inputs.zip` as well as `marriage_inputs_small.zip`, containing input files on which you are to benchmark your program as described below. The smaller inputs are fine for testing correctness, but you'll ultimately want to benchmark on everything in the larger archive. We will, in addition, test your program on other inputs that we do not provide you.

Input Specification

We provide an ASCII (8-bit) text file. The first line contains only a single integer, which may be multiple digits. That is the value of n: it is the number

of men (which is also the number of women). Every subsequent line contains $n + 1$ entries, separated by a space. For the first n lines after this first line, the first entry on each line is the name of a knight, and the rest of the entries are his order of preference for the ladies (we are only using a single name for each person, rather than first and last names, and there must be no duplicates). For the next n lines, the first entry on each line is the name of a lady, and the rest of the entries are her order of preference for the knights. Names may contain alphanumeric characters (they may look like real human names, or they may not).

Example:

```
2
Galahad Guinevere Elaine
Lancelot Guinevere Elaine
Elaine Galahad Lancelot
Guinevere Lancelot Galahad
```

Output Specification

You provide output on `STDOUT`. It must contain n lines, one per marriage. Each line has the knight's name, then a space, then the lady's name. We do not specify the order of the knights; we will sort your input and the correct answer in order to compare them.

Example:

```
Galahad Elaine
Lancelot Guinevere
```

If no input file is provided, or if the input format is not exactly as specified above, your program must exit with result code 1, and should print *nothing* on `STDOUT`. In python, this is achieved with

```
exit(1)
```

You must not return any other output on `STDOUT`. How, then, can you easily benchmark your program? We suggest one of two ways:

- On Linux or Mac OS X, use the `time` command to run your code from the command line.
- On any operating system, use the `timeit` module in Python. You can wrap the main body of your program in a call to `timeit.timeit` and then report the resulting time on `STDERR`.

If you do not understand what we mean by `STDOUT` and `STDERR`, please ask the course staff.

Part 2: Pandemic!

You are in a class of 25 students. Naturally, the seats in this class are arranged in a 5 x 5 grid.

Two students are neighbors if they are horizontally or vertically next to each other. For example:

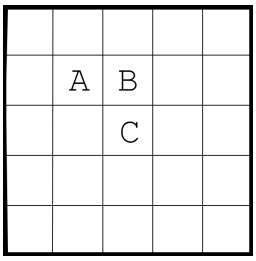
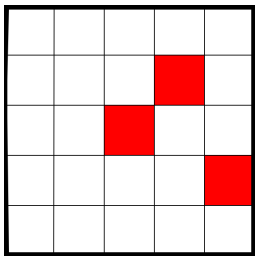
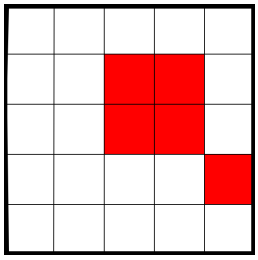
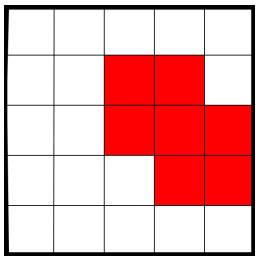
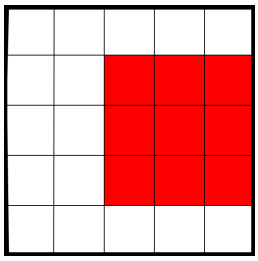
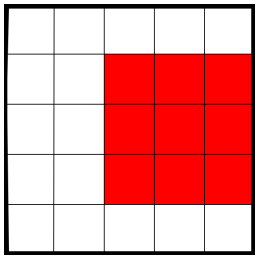


Figure 1: Day 1

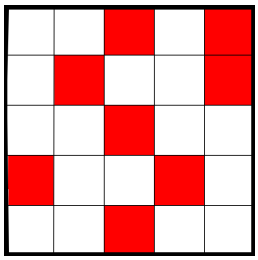
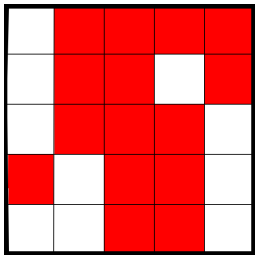
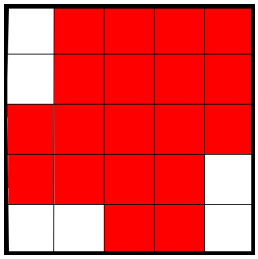
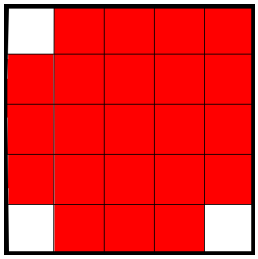
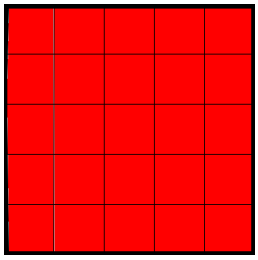
A and B are neighbors. B and C are neighbors. A and C are NOT neighbors.

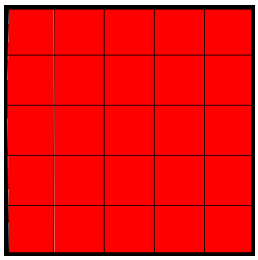
Some of your fellow students are infected by a mysterious virus (not SARS-COV2). An infected student never becomes healthy. Each day the infection spreads to a healthy student if that student has at least two infected neighbors. For example, let's start with the following configuration of infected (in red) students:

- Day 1: 
- Day 2: 

- Day 3: 
- Day 4: 
- Day 5+:  (the infection has stopped spreading)

For a second example, consider the following configuration of infected students:

- Day 1: 
- Day 2: 
- Day 3: 
- Day 4: 
- Day 5: 

- Day 6+  (every student is infected):

Answer as many of the following questions as you can. Explain your answers and the thoughts and processes that led you to those answers.

1. What is the maximum number of initially infected students such that, regardless of how they are placed in the classroom, at least one initially healthy student always remains healthy?
2. What is the minimum number of initially infected students such that there is some arrangement of that many initially infected students that will result in every student eventually becoming infected?
3. Can you arrange this minimum number of infected students in such a way that the infection never spreads to any healthy student?
4. How would your answers change if there were n^2 students in an n x n grid?
5. Does it matter if n is even or odd?
6. What *geometric* property about the set of infected students never changes as the days pass and the infection spreads?

Please submit your answers to the Pandemic problem as a separate PDF or text file along with your `marriage.py` on Gradescope.

Administrative stuff

Pair Programming

You will work with a partner for this entire assignment. Please refer to the pair programming policy in the syllabus. For this first assignment, we have assigned partners. For future assignments, you may choose your partner.

Lateness

Submissions will not be accepted after the due date, except with the intervention of the Dean's Office in the case of serious medical, family, or other personal crises.

Grading Rubric

Your grade will be based on four components:

- Functional Correctness of Marriage (25%)
- Design and Representation (10%)
- Invariant Documentation (25%)
- Benchmarking and Analysis (15%)
- Pandemic (25%)

Of these, only **Functional Correctness** will be autograded. Note that we are asking you to practice *defensive programming*: if you receive bad input, your program must exit with error code 1. We reserve the right to test your code with bad inputs. The idea here is that it is important not only to give the right answer with good inputs, but to not silently give the *wrong* answer with *bad* inputs. Doing so leads to insecure software.

Design and Representation is our evaluation of the structure of your program, the choice of representations (how do you represent a knight, for example? How do you represent a preference list? What **cost models** are appropriate given the access patterns?), and the use of appropriate (not excessive) comments. We will be relatively lenient in the Design and Representation grade on this first assignment, and progressively more demanding over the course of the semester.

Invariant Documentation is to force you to reason about the running time of the algorithm, as well as its correctness. Whenever you have a loop in the body of your algorithm, you should state whatever invariants hold. Consider what we discussed in class: **Initialization, Maintenance, and Termination**. So, at present, this shows up in the form of comments within your source code.

Benchmarking and Analysis will be relatively simple in this assignment. You should include a PDF in your upload to Gradescope, along with your program.

We provide you with four data files, each with a different value for n. The sizes are: 10, 100, 1000, and 5000 knights. You must time three runs on each of those

inputs, and compute the mean (average) time for each input. Then, plot those times against the input size. You may use any plotting tool you like; Excel is fine. Submit a PDF to Gradescope with your plot (along with your code), as well as an explanation of what you see. Does your plot bear out the $O(n^2)$ asymptotic complexity of Gale-Shapley? Why or why not? If not, can you come up with any explanation?

Tips and Pitfalls

- It's possible for a knight and a lady to have the same name. Your implementation must handle this.
- Be sure to have the **knights** propose first; if you have the ladies propose to the knights first, you will still end up with a stable marriage but it won't be the one expected by the autograder!
- Don't overcomplicate this assignment. My solution is 75 lines of code and involves no classes. I've seen solutions with a Knight class, Lady class, Proposal class, Marriage class... all for this one problem. Don't make more work for yourself.