Homework 1: Stable Matchings

Version 2 Posted 10/10 at 9 AM. We added hints to a few of the problems.

Due Date: The written parts of this assignment will be due at 11:59 PM on Friday October 14. As always, that means that the TAs will grade **every** written problem you submit to gradescope before that deadline. You will also be able to resubmit two written problems with every assignment, starting with homework 2. The coding problems may be submitted at any time through the last day of classes (Friday Dec. 9), but we strongly recommend you finish any you intend to do by the due date for the written assignment so you have time to do next week's problems next week.
Collaboration: You are allowed (and encouraged!) to discuss these problems at a high level with others. But, please read the full collaboration policy to ensure you are keeping your discussions at the right level, and remember to cite any collaborators.

Problems to Submit: We will count your 1 best mechanical question and your 3 best long-form questions.
We strongly recommend you skim all the problems in a section before attempting them. While we try to make problems in a given section of approximately equal difficulty, you may find some to be easier than others.

Mechanical Problems

1. Nobody's Happy [mechanical]

Is it possible for a stable matching to have no agent be matched to their first choice?
If yes, give an example **and** explain why your example works.
If no, prove why no such matching can exist.

Hint: There can be matchings other than the ones found by Gale-Shapley. This problem is harder (though definitely not impossible) if you only think about Gale-Shapley-discovered matchings.

2. Too Many Choices to Count [mechanical]

We proved in lecture that every stable matching instance has at least one stable matching, and that some stable matching instances have more than one. Design a family of stable matching instances such that the number of stable matchings is $\Omega(c^n)$ where c is a constant greater than 1.

By a "family" of instances, we mean you should give a description so that for any $n \geq 1$, we could write down the stable matching instance you want us to.

You must justify that your construction works, i.e. why an instance with n agents on each side has as many stable matchings as you claim.

We will give full credit for any $c > 1$ (with a proper argument).
Hint: for $n = 2$ you could use the instance on slide 11 of lecture 3. Try to generalize that example.

Long-form Problems

3. Proof by Contradiction [long-form, written]

You are one of the n riders being matched to n horses. You are one of k **popular riders**. Every horse has the k popular riders as their k top choices (though they might not agree on the ordering of the k riders).

Use a proof by contradiction to show that if the Gale-Shapley algorithm is run with riders proposing, you will be matched to one of your top k choices.

Caution: This problem looks superficially similar to worked exercise 1 in the Kleinberg and Tardos textbook – this problem is different!

Hint: Remember proof-by-contradiction always starts with the negation of the claim you want to prove. The negation of the implication $p \rightarrow q$ is $!p\&\&q$ (or $\neg p \wedge q$, in propositional logic notation). That's when the promise has been broken! So that's where you start. In this problem, the starting point should be "The Gale-Shapley algorithm, run with riders proposing matched you, a popular rider, to someone other than one of your top k choices."

4. Stable Matching Modeling [long-form, written]

In this problem you will practice performing a **reduction**. We'll spend a few weeks on reductions at the end of the quarter – your goal with a reduction is to use code written for a previous problem (say a library function someone else wrote) to solve a new problem **without editing or rewriting the library** (instead doing some pre- and post-processing to make the library solve the new problem).

You are given a function `StandardStableMatching`, which you will use to solve a new problem.

> `StandardStableMatching`
> **Input:** A set of k horses and k riders. Each horse gives a preference list of all k riders, and each horse gives a preference list of all k riders.
> **Output:** A stable matching among the k horses and k riders.

You have a set of n courses and n potential TAs. Each course would benefit from exactly one TA. But since these are TAs and courses, it won't be as simple as horses and riders. Each course can declare some (or none, or all) of the TAs as "unacceptable" – that is, they would rather have no TA (i.e., not be matched at all) than be matched to an unacceptable TA. Similarly, every TA can declare some (or none, or all) of the courses "unacceptable."
Every TA and course would prefer to be matched to any acceptable option than to be left unmatched.

In this context, call an assignment "stable" if:

- No TA is matched to a course they declare unacceptable.
- No course is matched to a TA they declare unacceptable.
- There is no non-matched TA-course pair that both declare the other acceptable and who both prefer each other to their current state.

Note that we do not require a perfect matching in this context to have a matching be stable – now that we have unacceptable pairings and differing numbers of applicants and jobs, we may leave some agents unmatched.

(a) Given n courses and n TAs, along with all of their preference lists and all their decisions as to whether other agents are unacceptable, describe how to use `StandardStableMatching` to find a stable matching.

For this problem, we do not care about the exact data structures you use to implement the lists – you may assume you start with 2-D arrays, or inverse arrays, or ordered lists, or any other reasonable representation. Similarly, you can assume that `StandardStableMatching` accepts whatever representation you prefer. We want you to focus on the big idea of how the library is useful to you, not the nitty-gritty details of converting between 2D arrays and ArrayLists.

(b) Explain why your output produces a stable assignment. (Our explanation is about 4-5 sentences)

(c) What is the running time of your algorithm? Assume that on an input with k riders and k horses, `StandardStableMatching` runs in time $\Theta\left(k^2\right)$. Justify your answer (our justification is two or three sentences).

5. Write the code! [long-form; coding]

The goal of this problem is to practice the kind of programming problems you'll see in this course and perhaps an interview. There will be some major differences between this course and 373:

- You'll have less starter code – we aren't training you to integrate your code into existing code bases, like we were in 373.
- You'll be allowed to use data structures libraries instead of always writing your own.

An autograder is provided on Gradescope. See the syllabus for how this score is converted into E/S/N/U letters.

In this question, you are asked to complete a method that takes in a stable matching instance and outputs an extreme matching most optimal to one particular side.

Input:

- A 2D array denoting the preferences of the horse. For example, arr[i][j] lists the j-th preferred rider of the i-th horse.
- A 2D array denoting the preferences of the rider.
- Whether if the matching provided should be horse-optimal or rider-optimal.

Output: An array denoting the matching in terms of the horse, namely, arr[i] contains the matched rider of the i-th horse.

Download the starter code StableMatching.java and complete the method marked "TODO". Then, submit StableMatching.java to the separate assignment on Gradescope. Do not submit it with your written portion.

A sample input is provided in the starter file, which you can use by running the Java class:

```
javac StableMatching.java && java StableMatching
```

You can also add in your own test cases in the main method.

If you don't have Java on your computer, click here to download it. Ask on Ed and Office Hours if you need help with setting up your computer!

Tips:

- Comment out any print statements before testing (our tests assume you won't print anything and will fail if you do).
- Implementing only the riders-proposing version will be enough to reach the "S" benchmark. Implementing both proposing will be required for "E".
- Our tests might not work like you expect. Just because you're passing a test, don't assume everything is perfect about that aspect of your code. We will directly check that you produce a stable matching, but we only indirectly check that you handle who proposes correctly.
- We'll give you the ability to test your code on gradescope later this week.
- Be sure to read and think carefully about what the input arrays mean. A very common bug last quarter was to misinterpret the meaning of the arrays (and therefore get incorrect code).

6. Real World: Application Review of Stable Matchings [long-form; written]

The goal of this exercise is for you to consider the effects of running algorithms in the real-world. This assignment is a mix of technical tasks (finding and applying theorems) and non-technical ones (considering tradeoffs between various real-world effects and groups). The technical aspects can be "right" or "wrong", but the non-technical aspects are unlikely to be simply "right" or "wrong" – we won't have to **agree** with the non-technical aspects of your analysis to consider them a good analysis. Our evaluation will be based on how well they connect to the technical aspects, as well as the depth of reasoning demonstrated.[1]

6.1. Application Review

Choose one of the following real-world uses of stable matchings:

- Medical Resident Matching (NRMP or programs in other countries)
- high school matchings (New York City, Boston, or other cities)
- Any other real-world application of stable matchings you can find
- A real-world scenario where stable matchings aren't currently used, but you think they could be (like a job market you're about to go on, for example).

6.2. Find a Theorem

We've covered a few theorems about stable matchings in lecture (e.g. proposer-optimality). In a reliable source, find a theorem about stable matchings we **haven't** covered in class.

- Copy-paste the theorem statement, the theorem number (or name, or some other unique identifier in that text), and cite your source.
- restate the theorem (in your own words) applied to horses and riders
- then state it as applied to your real-world application (e.g. 'doctors' and 'hospitals' or 'students' and 'schools').

Some places to look:

- Two-Sided Matching by Roth and Sotomayor
- Algorithmics of Matching Under Preferences by Manlove
- The Stable Marriage Problem: Structure and Algorithms by Gusfield and Irving
- Any other textbook or peer-reviewed paper

The first two books are available online through UW libraries (click the links). The third is available physically through UW libraries, but not online. For papers, you can usually find PDFs via google scholar (if they aren't available there, see if a librarian can help, or ask a staff member. Through UW library agreements, you should have access to just about every peer-reviewed paper written in the last 30 years).

Note that wikipedia/blog posts/etc. are **not** valid sources (though you may search through these places and then trace citations to find a reliable source).

[1] For example, if you say "Riders should propose to horses, because Gale-Shapley gives an advantage to people choosing between proposals" that's technically incorrect, because Gale-Shapley does the opposite.
If you say "Riders should propose to horses because Gale-Shapley gives an advantage to the proposing side, and I like riders more than I like horses" that's technically correct, but not well-thought out or justified.
If you say "Horses should propose to riders because Gale-Shapley gives an advantage to the proposing side, and riders are able to make an informed choice about whether to be a rider at this stable or another, while horses have no choice but to give rides at the stable they currently work at" that's correct technically, and well-justified (even if the staff-member grading believes horses aren't sentient and we should prioritize sentient species).

6.3. Consider the consequences

Identify two groups of people (or individuals) that are affected by this theorem in the context of the application you choose in part 1. State the consequences of the theorem for each of the groups. For each group also state whether you think it would be better for them to use a stable matching algorithm or a free-for-all market (like job markets in industry). These groups might be the horses and riders, or they might be subgroups within those groups, or even other people affected by the algorithm but aren't actually those being matched.

6.4. Proposing

We learned in class that Gale-Shapley can disadvantage the choosing side, but there is a nice property we haven't discussed. Gale-Shapley is a "truthful" algorithm for the proposing side. That is, it is if you know you will be on the proposing side, it will never be to your benefit to lie about your preference list (this statement only applies to the proposing side, not the choosing side).

When you're choosing whether to implement the stable matching algorithm in your new context. After some experimentation on old preference lists, you realize that getting a matching "in the middle" isn't going to be feasible (there are too many matchings in the middle to look through them and pick one fairly). Your supervisor gives you these options for disclosing your methodology to participants:

1. Announce before you receive preference lists that you will run Gale-Shapley with one side proposing.
2. Announce before you receive preference lists that you will run Gale-Shapley with the other side proposing.
3. Announce that you will flip a coin **after** receiving the preference lists. If its heads one side proposes, if it is tails the other side proposes.

Note that option 3 won't be "truthful" – you won't be able to tell people on either side that they won't benefit by lying. Consider the tradeoff between "truthfulness" and "fairness." Of those three options, which do you think is best in your scenario? Explain why in 3-4 sentences.

6.5. Summary

Based on what we've learned from class and the observations you've made so far, write a few sentences on whether you think stable matchings should be used in your scenario. (or if assignments should be made in a decentralized fashion, or some other model should be used to find an algorithm)