Homework 1: Stable Matchings

Due Date: The written parts of this assignment will be due at 11:59 PM on Friday January 19. 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 same requirement to use a resubmission slot applies to programming questions, though the tests on gradescope will remain accessible after the deadline, so you are encouraged to update the code on gradescope frequently until you get full credit.

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. You are welcome to submit more than this number (we will count your best in in each category).
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

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

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 $\Omega\left(c^n\right)$ 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 12 of lecture 3. Try to generalize that example.

Long-form Problems

3. Proof by Contradiction [written]

After some departmental restructuring, you were given the job of deciding how to allocate TA's to courses in your department. Due to budget cuts, every course is taking only one TA. All n TA's rank all of the n courses, and the course instructors for each of the n courses rank all of the n TA's. There are t **experienced TA's**. Every instructor has the t experienced TA's as their t top choices (though they might not agree on the ordering of the t experienced TA's).

Use a proof by contradiction to show that if the Gale-Shapley algorithm is run with TA's proposing, then every experienced TA will be matched to one of their top t 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` (that is, $p \wedge \neg q$, 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 "Suppose for the sake of contradiction, the Gale-Shapley algorithm, run with TA's proposing matched some experienced TA to a class outside of their top t choices."

4. Stable Matching Modeling [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 library function for solving the `StandardStableMatching` problem, which you will use to solve a "non-standard" variant of stable matching.

StandardStableMatching Solver

Solves an instance of "Standard" Stable Matching in an unspecified way (i.e. not necessarily Gale-Shapley).

Input: A set of k horses and k riders, where k is some positive integer. 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 m potential TAs and n courses. 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 "sub-stable" (to distinguish this new notion from "stable" as defined for standard stable matching) if the following 3 conditions hold:

1. No TA is matched to a course they declare unacceptable.
2. No course is matched to a TA they declare unacceptable.
3. There is no TA and course not currently paired to each other in the assignment that both declare the other acceptable and who both prefer to be with each other than remain in their current state. (*Note*: this includes TAs/courses that have no partner in the assignment.)

In this context, we do not require a perfect matching in order to have a matching be sub-stable; we can (and indeed may *need* to) leave some agents without a partner in the final assignment.

(a) You are given m TAs and n courses, each of which has:

- an ordered list of all agents that this particular agent considers acceptable in order of decreasing preferability (i.e. front of the list is most desirable)
- an unordered list (i.e. set) of all agents this particular agent considers unacceptable

Describe how to use the `StandardStableMatching` solver to find a sub-stable matching.

Note: Any submission that modifies Gale-Shapley instead of calling the given solver will receive a score of U. We're trying to have you practice using someone else's algorithm!

Hint: The instance of "non-standard" SM you've been given is not compatible with the standard SM solver, so consider augmenting both sides with additional "artificial" agents so that both sides have the same number of agents.

Make sure to specify preference lists for both real and artificial agents in enough detail to be able to back up your arguments in part (b).

(b) In this part, you will justify that your algorithm's output is indeed a sub-stable assignment.

(i) **Verify that the 1st condition holds.** (You may omit the verification of the 2nd condition since it will likely be very similar to the 1st.)

Hint: depending on how you approached part (a), you may need to use a proof by contradiction where you start by assuming that a TA x is paired in the final assignment with a course y that is unacceptable to x, then try to derive a blocking pair. This blocking pair should be x and some course b, where b will more than likely not be y.

Make sure to explain how the way you designed your preference lists justifies that x prefers b over y, and that likewise b prefers x over its current partner, call it a.

(ii) **Verify that the 3rd condition holds.**

Hint: Formally, (x, y) is a blocking pair if x prefers y over its current partner *and* y prefers x over its current partner.

Consider taking the logical negation of this definition to perform a proof by cases, then further subdivide those cases into subcases based on whether x/y's partner in the intermediate matching is acceptable, unacceptable, or artificial.

(c) What is the running time of your algorithm? Use the variables m and n as defined in part (a). You may 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 2–3 sentences.)

5. Write the code! [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 [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)