Homework 1: Stable Matchings

Due Date: Wednesday October 5th at 10 PM.

Be sure to read the grading guidelines and style guidelines on the webpage. Especially to see the suggested format for describing algorithms.

We sometimes describe how long are justifications or proofs are. These lengths are intended to help you estimate how much detail we're expecting; your proofs are allowed to be longer.

You are allowed (and encouraged!!) to collaborate with each other. Brainstorming is much easier to do in a group than alone! But you must follow the collaboration policy (which includes needing to write your submission on your own).

You will submit to gradescope; we will have a different box for each problem, so please give yourself extra time to submit.

1. Favorites [10 points]

Is it true that in every stable matching, there is some agent getting their first choice?

If so, give a proof (be sure to show the claim for every stable matching!).
If not, give a counter-example (be sure to justify that the matching is stable!).

2. Stable Matching Modeling [25 points]

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**. While you cannot alter the library, you will need to do some pre- and/or post-processing to make the library function appropriate for your use-case.

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

`BasicStableMatching`
Input: A set of n horses and n riders. Each horse has a preference list of all n riders, and each rider has a preference list of all n horses.
Output: A stable matching among the n horses and n riders.

Notice, you don't know how the `BasicStableMatching` function works. Maybe it's running Gale-Shapley. But maybe it isn't! And even if it is running Gale-Shapley, you don't know who is proposing. You do know, though, that the output is correct (i.e. it is stable) even if you don't know *which* stable matching is output.

Now, your task. You have a set of n job applicants and m jobs available. Both the job applicants and the companies offering the jobs *have standards*. Each job applicant can declare some (or none, or all) of the jobs as "unacceptable" – that is, they would rather have no job (i.e., not be matched at all) than be matched to an unacceptable job. Similarly, every job can declare some (or none, or all) of the applicants "unacceptable."
Every applicant and job would prefer to be matched to any acceptable option than to be left unmatched.

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

- No applicant is matched to a job they declare unacceptable.
- No job is matched to an applicant they declare unacceptable.
- There is no non-matched job-applicant 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 job applicants and m jobs, 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 n riders and n horses, `StandardStableMatching` runs in time $\Theta\left(n^2\right)$. Justify your answer (our justification is two or three sentences).

3. Someone will be the very best [25 points]

You and your friends are playing in a Pokèmon tournament. In the tournament, every player will have a battle with every other player (with exactly one player winning each matchup).

After all the battles, player u will **brag** if for every other player v:

- u beat v in their battle OR
- there is a player w such that u beat w and w beat v.

For example, in a tournament between Lorelei, Bruno, Agatha, and Lance, where:

- Lorelei beat Lance
- Bruno beat Lorelei
- Agatha beat Lorelei and Bruno
- Lance beat Agatha and Bruno

Lance can brag:

- Lance beat Agatha and Bruno directly and
- Lance beat Agatha who beat Lorelei.

(a) List all the people who can brag in the example above (no explanation required. [2 points]

(b) Give an example of a way battles could play out so that there is only one person who can brag. [3 points]

(c) Prove that no matter how the games play out, at least one player will brag. You **must** use induction for this problem, and you must "find the smaller one inside" in your inductive step. [20 points]