Homework 1: Stable Matchings

Be sure to read the grading guidelines and style guidelines in the Homework Guide. Especially 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 submit as you complete each problem.

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 more time on reductions later in 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 $2k$ agents in two groups of k agents each. Each agent has an ordered preference list of all k members of the other group,.
> **Output:** A stable matching among the $2k$ agents.

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 unmatched 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 `BasicStableMatching` 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 `BasicStableMatching` 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) Prove that you will output a stable assignment.

(c) What is the running time of your algorithm? Give a $\Theta()$ bound and justify with 1-3 sentences. To describe the running time, you may use n and m as defined above.
For the purposes of the analysis, assume that on an input with k riders and k horses, `BasicStableMatching` runs in time $\Theta\left(k^2\right)$ (but your final answer cannot include k, it's not a real real variable in this problem).

3. Overlapping Trees

Let $G = (V, E)$ be an undirected graph. Call a non-empty set S of vertices **travelable**[1] if for all $u, v \in S$ there is a path $u, w_1, ..., w_k, v$ where all $w_i \in S$ (i.e., you can get from u to v without ever using a vertex outside S; it's ok if $u = v$ or there are no w_i).

Call a graph **cool**[2] if for all sets A, B, C of travelable vertices, the following implication holds

$$[A \cap B \neq \varnothing \wedge A \cap C \neq \varnothing \wedge B \cap C \neq \varnothing] \to A \cap B \cap C \neq \varnothing$$

That is, if every pair of the travelable sets overlaps, then they all share at least one vertex.

We **strongly** recommend that you draw a few example graphs and overlapping travelable sets to make sure you understand the definitions and what the question is asking.

(a) Give an example of a graph that is not cool. Justify that it is not cool by giving three travelable sets that make the implication false. (Hint: read part b before trying this part). [5 points]

(b) Prove by induction that if G is a tree then G is cool.
You **must** use induction for this problem, and you must really use a smaller version of the problem inside the original problem in your inductive step. Your inductive step must be a direct proof (which means you may not use proof by contradiction for the main argument).
If your inductive step never considers the case that the implication is vacuous, you're probably missing something!

Hint: You can use the fact from section that every tree has a degree-one vertex.
Hint: If you want a shorter way to refer to the hypothesis of the implication, we say that "A, B, C pairwise intersect." [20 points]

[1] Sadly, not a real term.
[2] Very sadly, this is not a real term either. But all trees are cool.