Homework 1: Stable Matchings

Be sure to read the grading guidelines and style guidelines. 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. The Best of Times and The Worst of Times [10 points]

Give an example of a stable matching instance with 4 riders and 4 horses, with stable matchings M_1 and M_2 such that for some riders r and r'

- r prefers their match M_1 to their match in M_2
- r' prefers their match in M_2 to their match M_1

Note that this requires r and r' both change partners between M_1 and M_2

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 k horses and k riders. Each horse has a preference list of all k riders, and each rider has a preference list of all k horses.
Output: A stable matching among the k horses and k 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 lists of n doctors who are all applying for residency positions at m hospitals. The hospitals have a ranking over *individual* doctors, but each hospital h_i has a certain number of slots s_i available. There may be a different number of total slots than there are doctors applying, in which case some slots may go unfilled or some doctors may be unmatched.

In this context, call a pair (d, h) "blocking' if:

- d prefers h to its assigned hospital (or d is not matched) AND
- h prefers d to at least one of its assigned doctors (or h has at least one unfilled slot)

And call an assignment stable if each doctor is assigned to at most one hospital, each hospital h_i is assigned at most s_i doctors, and there are no blocking pairs. Note that we do not require everyone to be matched—now that we have potentially differing numbers of applicants and jobs, we may leave some doctors unmatched or slots unfilled.

Given n doctors, m hospitals, along with all of their preference lists and the s_i describing the maximum number of slots, describe how to use `BasicStableMatching` to find a stable assignment.

(a) Give a 1-3 sentence summary of what your algorithm will be (a summary makes it much easier for us to understand what you're doing).

(b) Describe how to find the matching. You can use whatever combination of English and pseudocode will let you be clearest.

(c) Give a 1-3 sentence summary of your intuition for your proof (a summary will make it easier for you to decide whether your claim is actually true! And helps us understand what you're doing).

(d) Prove that you will output a stable assignment.

(e) 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; you can also use $s = \sum s_i$.
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

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

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

(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 "find the smaller one inside" 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.