# 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, not just for the result of the Gale-Shapley (i.e. propose-and-reject) algorithm!
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 $2k$ agents, `BasicStableMatching` runs in time $\Theta\left(k^2\right)$ (but your final answer cannot include $k$, it's not a real variable in this problem).

## 3. Bragging rights [25 points]

Imagine that you and your friends are playing in a fencing tournament. In the tournament, every player will have a bout (fencing match) with every other player (with exactly one player winning each matchup).

After all the bouts, player $u$ has **bragging rights** if for every other player $v$:

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

In other words, $u$ can get bragging rights over $v$ by either defeating $v$ directly, or else (if $u$ lost to $v$) by defeating someone who defeated $v$ (so a one-hop transitive win).

For example, in a tournament between Atri, Weizi, Grant, and Paul, where:

- Atri beat Paul
- Grant beat Atri
- Weizi beat Atri and Grant
- Paul beat Weizi and Grant

Paul has bragging rights because:

- Paul beat Weizi and Grant directly and
- Paul beat Weizi who beat Atri.

(a) List all the people who have bragging rights in the example above (no explanation required).

(b) More generally with any number of players, describe how to set outcomes of the bouts so that there is exactly one person who has bragging rights.

(c) Prove that no matter the results of the bouts, at least one player has bragging rights. You **must** use induction for this problem. Remember that when you apply the inductive hypothesis to a smaller problem, that problem should depend on the larger problem you are given.