Homework Extra: Extra Problems

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, you should not take those estimates as hard length-limitations.

We are not imposing the usual length limit for these problems (problem 2 may have varying lengths for responses; we've tried to indicate about how much detail we're expecting; we think it's unlikely you'll need more than the page limit, but we won't restrict you).

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 have a different box for each problem, please give yourself time to submit.

1. Programming: make a change and get the choices [25 points]

We've given you a function `LIS`, which returns the **length** of the longest increasing subsequence of an array. Your task is to write code for a *variant* of the longest increasing subsequence problem.

Specifically, given an array, you should return the longest **geometrically** increasing subsequence. A subsequence is **geometrically** increasing if for chosen elements where $i < j$, we have $3A[i] < A[j]$. I.e. each successive element must be more than three times larger than the previous. For simplicity, you may assume there are no indices i, j such that $3A[i] = A[j]$ and that there are no repeated elements.

Your code will return an ArrayList containing **in order** the **elements** of a longest geometrically increasing subsequence. You therefore will need to modify the code given both to handle the geometric requirement AND to build the subsequence itself. (You're also free to start from scratch, but we think you'll find the starter code helpful).

You will submit your code to gradescope, where it will be autograded. You may submit until your code is correct (your score will be the score of your last submission before the deadline).

2. Real World: Application Review of Stable Matchings [25 points]

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]

2.1. 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)

[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).

- 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).

2.2. 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).

2.3. 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.

2.4. 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 (a few sentences should suffice here).

2.5. 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)