CS-3510 Algorithms, Spring 2026 Homework 1
McKell Roberts

Problem 1 (20 pts)

Prove or disprove:

a) (10 pts) For any instance of the stable matching problem, the following holds: In **every** stable matching, there is an **agent** (a company or an applicant) who gets their first (most favorite) choice.

False, we did this in class with the circular preferences didn't we? For example if we have A, B, and C and then we have X, Y, and Z with A preferring X, Y, Z and B preferring Y, Z, X and C preferring Z, X, Y and X preferring B, C, A and Y preferring C, A, B and Z preferring A, B, C no match up will have a single candidate with their first choice.

AY, BZ, CX. That is a stable matching and none of the partners are happy as they are all in their second choice. So no, an agent does not always get their first choice.

b) (10 pts) For any instance of the stable matching problem, the following holds: There **exists** a stable matching in which a company gets their first (most favorite) choice.

We pointed this out in class that it is almost the receivers that have all the power on whether they accept the proposal or not.

If all of the agents propose to their receivers first, at least one receiver will accept, with that being the most preferred proposal. The rest will trade up their preferential lists if they can. I think that made sense, that might have been a bit circular in my logic. But basically if all of P proposes to all of R, at least one R will have a P from all the Ps at the top of their list. They would never reject their preferred P and the Gale-Shapely algorithm guarantees a stable match so it should work.

Hint: If you believe the statement is true, write a proof that applies to emphevery instance of the stable matching problem. If you believe it is false, give a counterexample: specify an instance and stable matching(s), and briefly justify why the matching(s) are stable. One counterexample suffices.

Problem 2 (10 pts)

Prove that in every instance of the stable matching problem with n companies and n applicants, companies make at most $n(n-1)+1$ proposals in the Gale-Shapley algorithm.

If there are two Ps that are both matched with their least preferred R, then both of them were rejected by all the other, ($|R| = n$), n-1 Rs. Leaving one last switch possible between the two of them for their not preferred choice. The Ps could have all proposed to an R that favored them least, making that scenario above happen at most n times. SO there are n(n-1)

+ 1 proposals that are possible

Hint: First show that in the company-proposing Gale-Shapley algorithm, at most one company can end up matched with its last-choice applicant. Then use this to upper-bound the total number of proposals.

Problem 3 (10 pts)

Use induction to solve this problem: Given 2^k real numbers $x_1, \ldots, x_{2^k}$ such that $\sum_{i=1}^{2^k} x_i = 1$, show that:

$$\sum_i x_i^2 \geq \frac{\left(\sum_i x_i\right)^2}{2^k} = \frac{1}{2^k}.$$

Hint: You can use the following inequality: For any two real numbers a, b,

$$(a+b)^2 \leq 2a^2 + 2b^2.$$

Lets first show that if $k = 2$ then $2(x_1^2 + x_2^2) \geq (x_1 + x_2)^2$ so $\sum_{i=1}^{2k} = 1$ *then* $(x_1 + x_2)^2 = 1$ meaning $1/2 \leq x_1^2 + x_2^2$

Now lets consider $k+1$ so we are dealing with 2^{k+1} which is double 2^k. So lets pair the x's together, with $y_1 = x_1 + x_2$ *and* $y_2 = x_3 + x_4, ..., y_{2^k} = x_{2^{k+1}-1} + x_{2^{k+1}}$. Now we have 2^k y's, which fits our original equation of $\sum_{i=1}^{2^k} y_i = 1$

Problem 4 (10 pts)

Let $I = (P, R)$ be an instance of the stable matching problem. Suppose that the preference lists of all $p \in P$ are identical, so without loss of generality, p_i has the preference list $[r_1, r_2, \ldots, r_n]$. Prove that there is a unique solution to this instance. Describe what the solution looks like, and why it is the only stable solution.

IF the proposers all have the same list, then it's really the receivers making the decisions. r_1 will have it's favorite match as it's everyone's favorite, so that happy pair can then be removed from the list. Making r_2 the new first choice of all that remains, they get their first pick of what remains. AND on and on it goes until r_n and the last p_i is forced together as r_n was the least desired.

Note: Showing that the solution is the one found by the Gale-Shapely algorithm is not sufficient, as there could be other solutions.

Problem 5 (15 bonus pts)

Implement the Gale-Shapley algorithm to solve the Stable Matching Problem using $O(n^2)$ time complexity. You are free to write in any programming language you like. Make sure that you test your algorithm on small instance sizes, where you are able to check results by hand. Run your algorithm on the following testing instance of size $n = 4$ (define the algorithm as a function and give preference matrices as function parameters). Show **the resulting matching** that is found by your function, along with **list of intermediate proposals** performed by the algorithm. **your source code file as well**.

- **Proposers (P):** $P = \{1, 2, \ldots, n\}$
- **Receivers (R):** $R = \{1', 2', \ldots, n'\}$
- **Preference profiles:**
  - Each proposer has a ranked list of receivers.
  - Each receiver has a ranked list of proposers.

Function definition: `gale_shapley(P_pref, R_pref)`

Input:

- Two preference matrices:
  - `P_pref[p][i]`: i-th preferred receiver of proposer p.
  - `R_pref[r][i]`: i-th preferred proposer of receiver r.

Output:

- `match_P[p]`: The receiver matched to proposer p.
- `match_R[r]`: The proposer matched to receiver r.

Testing Input:

$$\texttt{P\_pref} = \begin{bmatrix} 3 & 2 & 4 & 1 \\ 1 & 2 & 4 & 3 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

$$\texttt{R\_pref} = \begin{bmatrix} 1 & 3 & 2 & 4 \\ 3 & 1 & 4 & 2 \\ 4 & 3 & 2 & 1 \\ 3 & 4 & 2 & 1 \end{bmatrix}$$

Note: You may refer to the lecture slide for efficient implementation. You can switch to 0-based indices if necessary.

```
def galeShapley(pPref, rPref):
  n = len(pPref)

  #Initially all p in P and r in R are free
  free = list(range(n))
  nextProposal = [0] * n
  matchP = [-1] * n
  matchR = [-1] * n

  #Precompute ranking of proposers for each receiver
```

```
    rank = []
    for r in range(n):
        ranking =
        for i, p in enumerate(rPref[r]):
            ranking[p - 1] = i
        rank.append(ranking)
    proposals = []

    #While there is a proposer p who is free and hasnt proposed to every r
    while free:
        p = free.pop(0)
        r = pPref[p][nextProposal[p]] - 1
        proposals.append((p + 1, r + 1))
        nextProposal[p] += 1

        #If w is free then (p, r) become engaged
        if matchR[r] == -1:
            matchR[r] = p
            matchP[p] = r
        else: #Else r is currently engaged to p
            current = matchR[r]

            #If r prefers p to p then p remains free
            if rank[r][p] ¡ rank[r][current]:
                matchR[r] = p
                matchP[p] = r
                matchP[current] = -1
                free.append(current)
            else: #Else r prefers p to p, (p, r) become engaged, p becomes free
                free.append(p)

return matchP, matchR, proposals
(1,3),(2,1),(3,2),(4,4)
```