Introduction to Algorithms	**Problem Set 1 (3 problems)**
CS 4820 Spring 2023	**Due: Feb 2, 2023 (Thursday) 11:59 pm**

Hand in your solutions electronically on Gradescope (except for coding problems, which should be submitted as indicated in the question). Your submission should be typeset (except for hand-drawn figures). Collaboration is encouraged while solving the problems, but:

1. list the NetID's of those in your group;
2. you may discuss ideas and approaches, but you should not write a detailed argument or code outline together;
3. notes of your discussions should be limited to drawings and a few keywords; you must **write up the solutions and write code on your own**.

Remember that when a problem asks you to design an algorithm, you must also prove the algorithm's correctness and analyze its running time, i.e., the running time must be bounded by a polynomial function of the input size.

(1) (10 points) In this question we consider the "original" form of the stable matching problem, which was solved by the National Resident Matching Program (NRMP). See exercise 1.4 in the text for a full problem description. Suppose there are n students and m hospitals, with the number of students greater than *or equal to* the total number of slots. Each student has a complete preference list of all the hospitals, and each hospital has a complete preference list of all the students.

While the NRMP uses resident-optimal matchings, in this question we ask you to produce hospital-optimal matchings. Implement an algorithm in Java, C, C++, OCaml, or Python to compute a maximal, hospital-optimal, stable matching. **We strongly recommend using Java, C, or C++**, and will not guarantee that a full score is attainable if you use OCaml or Python. Your program should terminate within 5 seconds on all inputs in the autograder environment.

The input format (passed via `stdin`, terminated by a newline) is as follows:

- a line containing two positive integers: m, the number of hospitals and n, the number of students.
- m lines containing one positive integer each, the capacity c_i of each hospital for $1 \leq i \leq m$.
- m lines each containing a permutation of $\{1, \ldots, n\}$, the preferences of each of the hospitals.
- n lines each containing a permutation of $\{1, \ldots, m\}$, the preferences of each of the students.

Your program should output, to `stdout`, n lines containing the hospital for each student, in order. For unmatched students, output `0`. Your output should be terminated by a newline.

Here are some input bounds:

- $mn \leq 20\,000\,000$.
- At least 20% of test cases will have all hospitals of capacity 1, and $m = n$.

Here are some hints.

1. Read section 2.3 to see an implementation of Gale-Shapley that takes $O(n^2)$ time.
2. We hope that similar to Gale-Shapley, we can use a proposal-based algorithm. Since there are m hospitals and n students, and Gale-Shapley can be implemented with a runtime of $O(n^2)$, ideally we would have a runtime of $O(mn)$.

3. One way to *reduce* this problem to the normal stable matching problem is as follows:

   (a) For each hospital h_i with a capacity of c_i, create c_i mentors. Each mentor will represent one of the residency slots at this hospital and will have the same preferences as the hospital.

   (b) Create additional $n - \sum c_i$ "last-choice mentors" not associated with any hospital.

   (c) Rewrite the students' preference lists so that instead of listing hospitals, they list mentors. Put the "last-choice mentors" at the end of the students' preference lists.

   (d) Find a stable matching of the n students to the n mentors with Gale-Shapley.

   The runtime of this algorithm as stated is not $O(mn)$, but it's a good starting point.

Once you've finished your implementation, log into the autograder and submit to the problem named **hospitals** for your course number. The autograder will queue your submission to run on a small number of *public test cases* and report the results to you. However, your score will be largely determined by the number of *private test cases* which your program solves correctly after the submission deadline has passed. We recommend that you verify immediately that your code passes all the public test cases, as it is unlikely to pass any private test cases otherwise.

(2) (10 points) In this problem, we study the Gale-Shapley stable matching algorithm in an adversarial setting. Assume that the adversary knows the preference list of every hospital and resident, and can do the following: it can change *exactly one* preference list (of a hospital or a resident) and substitute it with an arbitrary preference list of the adversary's choice. The Gale-Shapley algorithm is then executed on this modified input.

Let $\mathcal{M}'$ denote the resulting matching, and let $\mathcal{M}$ denote the matching produced by running the Gale-Shapley algorithm on the original, unmodified input. We say that the attack is *bad for hospitals* if every hospital strictly prefers their matched resident in $\mathcal{M}$ to their resident in $\mathcal{M}'$. Similarly, we say the attack is *bad for residents* if every resident prefers their matched hospital in $\mathcal{M}$ to the match in $\mathcal{M}'$. (Note that "prefers" refers to the actual preferences, not the preference list which may have been substituted by the adversary.)

For every $n > 1$, prove the following:

(a) Show that there exists a stable matching instance with n hospitals and n residents such that an adversary can be bad for hospitals by changing one hospital's preference list.

(b) Show that there exists a stable matching instance with n hospitals and n residents such that an adversary can be bad for residents by changing one hospital's preference list.

(3) (10 points) There are n soccer teams that have practice scheduled in n different fields on a particular week. There are m time slots during the week, where $m > n$. Each team i has a fixed *schedule* which gives, for each of the n fields, the time slot in which team i gets to practice on that ground. This, in turn, defines a schedule for each field j, giving the time slots in which different teams practice on field j. The schedules have the property that

- each team practices on each field exactly once,
- no two teams practice on the same field in the same time slot, and
- a team cannot practice on multiple fields in the same time slot.

Now, it so happens that the teams decide that moving around throughout the week can be tiresome, and they come up with the following scheme. Each team i will pick a *distinct* field j. At the end of i's practice session on field j, team i will continue to use field j for the remaining of their practice sessions the week (and not move around for practice), and at this point field j will cancel its remaining reservations (by other teams).

The crucial thing is, the teams want to plan this cooperatively so as to avoid the following *bad situation*: some team i whose schedule has not yet been truncated (and so is still following their original schedule) shows up for a practice session on a field j that has already cancelled all reservations for the week.

Give an efficient algorithm to arrange the coordinated selections of the teams and fields so that this scheme works out and the *bad situation* described above does not happen.

Example: Suppose $n = 2$ and $m = 4$; there are teams t_1 and t_2, and fields f_1 and f_2. Suppose t_1 is scheduled to practice at f_1 in slot 1 and practice at f_2 in slot 3; t_2 is scheduled to practice at f_1 in slot 2 and f_2 in slot 4. Then the only solution would be to have t_1 to select f_2 and t_2 to select f_1; if we match t_1 to leave with f_1, then we'd have a bad situation in which f_1 has cancelled all reservations after the first slot, but t_2 still shows up for practice at f_1 at the beginning of the second slot.