Introduction to Algorithms	**Problem Set 1 (3 problems)**
CS 4820 Spring 2022	**Due 11:59pm Thursday, February 3**

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) Stable Matching with Forbidden Pairs (10 points) Recall the stable matching problem from lecture. In this problem, we will consider a slight modification to it involving forbidden pairs. You are given n proposers and n respondents, a preference list for each proposer and respondent of all members of the opposite type, along with a list of "forbidden" proposer-respondent pairs. Define a *valid* matching as a perfect stable matching containing no forbidden pairs. Give an implementation of an algorithm that outputs a valid matching such that the resultant matching is best for the proposers, or states that no such valid matching exists, and runs in $O(n^2)$ time. Feel free to reference pages 19–22 of the textbook for a more thorough description and analysis of the forbidden pairs problem, and pages 46–47 of the textbook for an explanation for implementing the Gale-Shapley algorithm. See the Programming Assignment Instructions on canvas (posted under Modules > Resources) for guidelines on what languages you can use, and how you can use the autograder to test your code on public test cases.
Input / output formatting and requirements. Your algorithm is to read data from **stdin** in the following format:

- The first line has two integer numbers, $2 \leq n \leq 2,000$ and $0 \leq m \leq \min(n^2, 50,000)$. Proposers and respondents are labeled with numbers $0, 1, \ldots, n-1$.
- In each of the next n lines, we are providing the preference list of a proposer. The i'th line of this part of the input is the preference list of proposer $i-1$ (the first respondent in the list is the most preferred and the last respondent is the least preferred).
- In each of the next n lines, we are providing the preference list of a respondent. The i'th line in this part of the input is the preference list of respondent $i-1$ (the first proposer in the list is the most preferred and the last proposer is the least preferred).
- In each of the next m lines, we are providing a description of a forbidden pair. Each line consists of two numbers a, b, where a is the index of the proposer and b is the index of the respondent in the forbidden pair.

Your algorithm should output data to **stdout**. Be sure to terminate your input with **a single newline character (\n)** .

- The first line should contain either the string "Yes" if a valid matching exists, or the string "No" if no valid matching exists. These strings must be printed exactly as shown.

- If no valid matching exists, then no further output (besides the trailing newline) is required. Otherwise, output the valid matching in the following way: for each new line i, output the number of the **proposer** that the i'th **respondent** is matched with.

Example.

When the input is:

```
2 1
0 1
1 0
0 1
1 0
0 0
```

There are two proposers and two respondents, along with one forbidden pair. Thus, there does not exist a valid matching in which proposer h_0 is matched with respondent r_0. Due to this, an algorithm might run as follows:

1. proposer h_0 proposes to respondent r_0 and gets rejected,
2. proposer h_0 proposes to respondent r_1,
3. proposer h_1 proposes to r_1.

We can see that h_0 does not have any more respondents left, and yet is still not matched; thus, no valid (perfect, stable, and constraint-satisfying) matching exists and the output should be

```
No
```

In contrast, if the input was

```
2 1
0 1
1 0
1 0
0 1
1 1
```

we can, in fact, construct a valid matching that satisfies the constraints $((h_1, r_0), (h_0, r_1))$. Thus, your algorithm should find this perfect matching and output

```
Yes
1
0
```

as each line is the number of the proposer that the i'th respondent is matched with.

A note about running times.

Suppose myList is a linked list with n elements, where i and j are two of the elements in myList, and consider the running time of the following code snippet (which happens to be Java, but the same applies for other languages).

```
if (myList.indexOf(i) < myList.indexOf(j))  {
         writer.println("Yes");
} else {
         writer.println("No");
}
```

The running time is *not* $O(1)$. Make sure you understand this, so you don't make a common mistake in implementing the Gale-Shapley algorithm that leads to a running time of $O(n^3)$ instead of $O(n^2)$ (which will make a big difference for the larger test instances where $n \approx 2,000$).

(2) Matching with Partial Lists (10 points) We studied the stable matching in class assuming that each side has a full preference list, and that the number of hospitals and residents is the same. However a natural scenario is that residents are limited to apply only to a small set of hospitals. To make this concrete, assume that we have the following (somewhat simplified) arrangement.

- There are n residents and a set of m hospitals. We will assume that each applicant is asked to list k hospitals that they are interested in, and lists them in the order of preference.
- Each hospital gets the full list of applicants and ranks them in order of preference. Each hospital should get matched to exactly one resident (we assume for the sake of simplicity that every resident applies for the same specialty, and that they are all competing for the same position).
- In this situation, we cannot expect to match all residents to hospitals. We will extend the definition of a stable matching to be a matching that satisfied the following conditions which we'll call weak stability:
  - For all matched pairs (r, h) the hospital h was listed on r's preference list,
  - for any pair (r, h) of resident r, and hospital slot h on r's preference list, if the pair (r, h) is not part of the matching, then one of the following must hold:
    - h is assigned to a different resident r' and h prefers r' to r
    - r is assigned to a different hospital h' and r prefers h' to h.

This is very similar to the traditional stable matching problem, but some hospitals as well as some prospective residents may remain unmatched.

Show that there is a weakly stable matching for any set of prospective residents and hospitals and any preference lists, and give an algorithm to find one in at most $O(nm)$ time.

(3) Interview Scheduling (10 points) Consider the following scenario: n students are flown out to California for a day of interviews at a large software company. The interviews are organized as follows. There are m time slots during the day, and n interviewers, where $m > n$. Each student s has a fixed *schedule* which gives, for each of the n interviewers, the time slot in which s meets with that interviewer. This, in turn, defines a schedule for each interviewer i, giving the time slots in which i meets each student. The schedules have the property that

- each student sees each interviewer exactly once,
- no two students see the same interviewer in the same time slot, and
- no two interviewers see the same student in the same time slot.

Now, the interviewers decide that a full day of interviews like this seems pretty tedious, so they come up with the following scheme. Each interviewer i will pick a *distinct* student s. At the end of i's scheduled meeting with s, i will take s out for coffee at one of the numerous local cafes, and they'll both blow off the entire rest of the day drinking espresso and watching it rain.

Specifically, the plan is for each interviewer i, and their chosen student s, to *truncate* their schedules at the time of their meeting; in other words, they will follow their original schedules up to the time slot of this meeting, and then they will cancel all their meetings for the entire rest of the day.

The crucial thing is, the interviewers want to plan this cooperatively so as to avoid the following *bad situation*: some student s whose schedule has not yet been truncated (and so is still following his/her original schedule) shows up for an interview with an interviewer who's already left for the day.

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

Example: Suppose $n = 2$ and $m = 4$; there are students S_1 and S_2, and interviewers I_1 and I_2. Suppose S_1 is scheduled to meet I_1 in slot 1 and meet I_2 in slot 3; S_2 is scheduled to meet I_1 in slot 2 and I_2 in slot 4. Then the only solution would be to have I_1 leave with S_2 and I_2 leave with S_1; if we scheduled I_1 to leave with S_1, then we'd have a bad situation in which I_1 has already left the building at the end of the first slot, but S_2 still shows up for a meeting with I_1 at the beginning of the second slot.