[redacted]

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. **Implement the Gale-Shapley algorithm**. In this problem we consider the classical Gale-Shapley algorithm. Given n proposers and n respondents, and a preference list for each proposer and respondent of all members of the opposite type, give an implementation of the Gale- Shapley stable matching algorithm with proposers proposing that runs in $O(n^2)$ time, as explained on page 46 of the textbook. 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 one integer number, $2 \leq n \leq 2,000$. 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 ith 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 ith 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).

Your algorithm should output data to `stdout` in two lines, the first line containing number a_1 and the second line containing number a_2, **followed by a single newline character (`\n`)** , where the desired values of a_1 and a_2 are described below:

- a_1 should be the number of the proposer that respondent 0 is matched to in your stable matching.
- a_2 should be calculated as follows: suppose now that proposer 0's first proposal is always rejected. Let us call the first proposer h_0 and denote by r the respondent that is first on h_0's list. Then r rejects h_0's proposal, even if r is unmatched or would prefer h_0 to their current match. (All other behavior is exactly the same as in the classical algorithm). This changed behavior of r can result in a few different outcomes:
  1. respondent r does not get matched at all.
  2. respondent r is part of an instability.
  3. respondent r is matched to the same partner or a better partner than in the matching resulting from the standard algorithm.

  a_2 should be equal to the number of the case that occurred.

Example.

When the input is:


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


There are two proposers and two respondents, and in the classical Gale-Shapley algorithm runs as follows:

(a) proposer h_0 proposes to respondent r_0,
(b) proposer h_1 proposes to respondent r_1,

and a stable matching is formed consisting of (h_0, r_0), (h_1, r_1). So the answer for part (a) is 0. For part (b) the first proposal by proposer h_0 is rejected, so this algorithm proceeds as follows

(a) proposer h_0 proposes to respondent r_0, and gets rejected (due to the changed rule),
(b) proposer h_0 proposes to respondent r_1,
(c) proposer h_1 proposes to respondent r_1 and gets rejected as r_1 prefers h_0,
(d) proposer h_1 proposes to respondent r_0.

This results in the matching (h_0, r_1), (h_1, r_0), which is stable, and respondent r_0 has an improved partner, so the answer to part (b) is 3. Thus, your output for this test case should look like:

```
0
3
```

In contrast, if the input was

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

the answer to part (a) is the same, but the result for part (b) is that r (which is r_0 again) will remain unmatched, so the answer to part (b) is 1, and thus your output for this test case should look like:


```
0
1
```


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. **Stable Matching with Identical Preferences**

   (a) Let us say that a hospital h in a stable matching instance is the top-ranked hospital if all residents rank h at the very top of their list; similarly, call a

resident r the top-ranked resident if all hospitals rank r first. Note that an instance does not have to have a top-ranked hospital or top-ranked resident. Prove that if there exists a top-ranked hospital and a top-ranked resident, then they must be matched to each other in any stable matching.

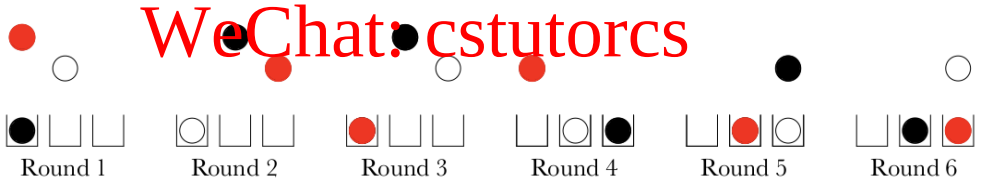
(b) A stable matching instance has *identical preferences* if all hospitals have the same ranking of residents, and all residents have the same ranking of hospitals. Use part (a) to prove **by induction** that every stable matching instance having identical preferences has a unique solution. Format your answer with the induction hypothesis, base case, and induction step clearly identified.

(c) Give an example of an instance of the stable matching problem that does not have identical preferences, and does not have a top-ranked hospital or top-ranked resident, yet it has a unique solution. In your answer, you should describe the preferences of each hospital and resident, identify the unique stable perfect matching, and explain why there are no other stable perfect matchings.

(d) Design and analyze a polynomial-time algorithm to decide if a stable matching instance has a unique solution.

3. **Gotta Catch 'Em All.**

In the game Gotta Catch 'Em All there are m balls (each having a different color) and $n \geq m$ tubes. Each tube can hold one ball at a time, and the game proceeds in $k > n$ rounds during which the balls "jump" between the different tubes. In each of the k rounds, each ball is either in a tube (and it is the only ball in that tube at that time) or it is in the air. Here is an example with $m = 3$ balls, $n = 3$ tubes and $k = 6$ rounds.



To play the game, the player first observes what happens over the course of the k rounds. The exact same k rounds then repeat again, and now the player can *close* zero or more tubes in each round to catch the ball that is in the tube. Once a tube is closed, the ball is caught there and won't move anymore; no other ball can visit the tube in this case, so if another uncaught ball tries to land in the tube, the player loses. The goal of the game is to catch all balls in the next k rounds.

In the example above, tube 1 must catch the red ball in round 3 (because otherwise tube 1 either does not catch anything or it closes before round 3, and then in round 3 the red ball tries to land in a closed tube and you lose). We can either catch the white and the black ball in tubes 2 and 3 in round 4, or we can catch the white ball

in tube 3 in round 5 and the black ball in tube 2 in round 6.

(a) Suppose that for each ball, there is at least one round in which the ball is in a tube (otherwise, you can obviously not catch all balls and win the game). Give an example that shows that you still may not be able to catch all balls.

(b) Now suppose that for each ball and each tube, there is exactly one round in which the ball visits that tube (this is true for the example above). Prove that all balls can be caught in this case, and give an algorithm that tells you in which round to close each tube.