COP 3503 Homework #5: Short Codes

Filename: codes.java
Time Limit: 4 seconds (per input case)
Standard Input, Standard Output

Many drugs have very long names that are hard to pronounce and take doctors a long time to write down. In an effort to speed up the time it takes for doctors to write prescriptions, the MAA has suggested that each drug name be mapped to a shorter code.

Doctors are great at memorizing codes, but it's easier if the code for a drug is a substring within the name of the drug. To that end, the MAA has produced a list of $\boldsymbol{m}$ potential codes for $\boldsymbol{n}$ ($\boldsymbol{n} \le \boldsymbol{m}$) drugs. The goal is to match each drug to a unique code (an injective function) such that each code is a substring of the drug it represents.

For this homework assignment you'll have three tasks:

(a) Determining IF it's possible or not to make such an assignment.

(b) If it is possible, determine any such valid matching.

(c) If it is possible, finding the first lexicographical mapping that is valid. In particular, the drugs will be given in a particular order. One mapping comes before another mapping lexicographically, if the first drug is mapped to a code that comes earlier in alphabetical order. (In the example below, lisinopril comes first. It could be mapped to the codes "no", "sin", or "il". Of these, "il" comes first alphabetically. Thus, is there exists some mapping that works with lisinopril being mapped to "il" this is preferred over mappings that map lisinopril to other codes. Between two mappings, one is lexicographically first if the first drug in the list that is mapped differently is mapped to a code that comes earlier alphabetically.

Let's take a look at an example:

List of Drugs	List of Abbreviations
lisinopril	bu
ibuprofen	pro
acetaminophen	no
amoxicillin	sin
naproxen	il
	tam

In this example, a mapping is possible:

```
lisinopril    → sin
ibuprofen     → bu
acetaminophen → no
amoxicillin   → il
naproxen      → pro
```

But, this isn't the first lexicographical mapping. That would be:

```
lisinopril    → no
ibuprofen     → bu
acetaminophen → tam
amoxicillin   → il
naproxen      → pro
```

The Problem:

Given a list of ***n*** drugs and ***m*** potential codes for those drugs, determine if there exists a way to match each drug to a different code such that each drug has a code that is a substring of the drug name. If it's possible, determine one possible matching, and for maximal credit, determine the first lexicographical matching.

The Input:

The first line of input contains two integers, ***n*** ($1 \le n \le 100$), representing the number of drugs for the input case and ***m*** ($n \le m \le 120$), the number of potential codes for the input case.

The next ***n*** lines each contain the name of a single drug, in the order to be considered for the matching. Each of these names will be **unique** strings of in between 10 and 30 lowercase letters, inclusive.

The following ***m*** lines each contain the name of one code. Each of these codes will be **unique** strings of in between 2 and 9 lowercase letters, inclusive.

The Output (for Task 1)

Output a single string, either "yes" or "no" on a line by itself depending on whether or not its possible to map each drug to a different code in the given list.

The Output (for Tasks 2/3)

If the first line of output for Task 1 is "yes", then you'll output ***n*** more lines as follows:

The first line being the code the first input drug is mapped to, the second being the code the second input drug is mapped to, and so forth.

If the mapping you output is the first lexicographical one, then full credit will be given.

Sample Input	**Sample Output**
5 6 lisinopril ibuprofen acetaminophen amoxicillin naproxen bu pro no sin il tam	yes no bu tam il pro
5 5 lisinopril ibuprofen acetaminophen amoxicillin naproxen nap pro no fen cil	no
5 5 lisinopril ibuprofen acetaminophen amoxicillin naproxen nap pro no li il	yes il pro no li nap

<u>Implementation Requirements</u>
This problem can be solved by reading in the input, forming a network flow graph, then running a max flow algorithm on that graph. (For the third task, the algorithm has to be run multiple times.)

It is expected that you use the following posted code with the two network flow algorithms presented in class:

FordFulkerson.java

or

Dinic.java

While you may develop your own network flow code, doing so is quite time consuming, so the recommended path is to use the posted code. (This is courtesy of the UCF Programming Team circa 2016.)

If you only want to complete the first task, then no editing of this code is required and all you have to do is call the methods in the class without ever accessing any information in the class from outside of it. If you want to complete the second and/or third task, you'll either have to access the instance variables of your FordFulkerson or Dinic object OR add methods to one of those classes. This editing will require you to understand (to some extent) what the code is doing and what each variable is storing and when. This is perhaps more similar to a real-world task where you frequently use others' code and may have to modify it some.

<u>Rough Grading Breakdown</u>
Code points will determine 40 points out of 100.

There will be 10 test cases for task 1 – the output for each of these will be worth 2 pts per test case. **<u>Credit will only be given if the proper algorithm is used. Namely, by sheer luck one is expected to match half of the outputs (one could just write a one line program that outputs "no" all the time. In these cases, even if the output matches, no credit will be given.</u>**

There will be 10 test cases for tasks 2/3 (each of which have more than one valid matching) – you earn 2 pts for any valid matching, and 4 pts for the lexicographically first matching.

Thus, a perfect score for completing Task 1 is 60/100, Tasks 1 and 2 is 80/100 and all three tasks is 100/100. Note that task 1 is quite easy (since you're using pre-written code), and task 2 is moderate (much less original code than programs 2, 3 and 4). Task 3 is quite challenging, so the expectation is that most students will get around 80 on this assignment.

What To Submit

For this assignment, please submit a single Java program named `codes.java` which solves the posted problem. ***In your header comment, state whether you solved just task 1, or tasks 1 and 2, or all 3.***

Note: Your program should either just output yes or no, OR for yes cases it should output yes followed by one single matching on n lines. We'll judge both tasks 2 and 3 based on the single matching you output. Your program will be tested on different input cases for Tasks 1 vs. Tasks 2 and 3.