Let's Do Lunch
A short programming exercise

Peggy Piranha and Samuel Amherst Salmon III (Sam) are old friends looking to get together for lunch. They haven't seen each other in ages, but are keen on crossing paths for lunch. But where could they meet? They're flexible on the ultimate destination, but there are few constraints.

Peggy and Sam live in a fairly complicated watershed, with many rivers running together and splitting apart. Sam has to keep moving upstream for spawning season. Peggy has dinner plans with a killer whale later in the day, and has to keep moving downstream. Fortunately, we have a map given to us by the Piscine Post Office, with each junction marked with an address. Each segment of river is listed as pairs of addresses, from upstream to downstream. Given the map and a list of possible starting locations for Peggy and Sam, give the list of locations where they could meet. Oh, and there are a few places they must avoid passing through, with waterfalls, rapids, and other unpleasantness. Your job is to list the possible addresses where they could meet, one per line and sorted alphabetically by address.

Input/Output
Input and output are simple text, via standard input and standard output, respectively. Input will be provided in the format shown in the example below: the list of address pairs forming the map, followed by the list of addresses to avoid, the list of possible starting locations for Peggy, and finally the list of possible starting locations for Sam.

Sample Input	Sample Output	Map (for reference)
Map: a1 b1 a2 b1 a2 b2 a2 b3 b1 c1 b2 c2 b2 c3 b3 c3 Avoid: b2 Peggy: a2 Sam: c2 c3	a2 b3 c3	

Guidelines, corner cases, and other details
- Assume that the input is always well-formed - no need to guard against bad input.
- Address labels are a string which can have any character other than a space or newline. For example, a1 or ThisIsAnAddress. Don't think too hard about it, it's just a string to uniquely identify the node. Address names are always unique, the same word always indicating the same location.
- If there are multiple locations to avoid, or multiple starting locations, they will be listed on a single line separated by spaces, e.g.
    a1 a2 a3
- If there are no locations to avoid, there will be a blank line, e.g.

```
Avoid:

Peggy:
a1
```

- Peggy and Sam will each always have at least one possible starting location.
- The output should be a list of locations, one per line, sorted alphabetically. If there are no possible meeting locations, there is no output either.
- Do not print anything other than the expected output - no headers, status messages or anything else (it would interfere with automated testing).
- If a starting location appears in the locations to avoid, the location must be avoided, and should not be passed through.
- Cycles in the map are possible. It is a complicated watershed, after all.

Submitting an answer
To submit an answer, write a solution in Java and send us the source code. You may use Java's standard libraries, e.g. java.util.Set.

Evaluation
Once received, we'll run your solution against our test cases and review your source code.

First and foremost, we look for code which is readable. There is more than one way to solve this exercise, but the best solutions strike the right balance between conciseness and clarity. Your choice of algorithm will likely have more impact than your comments, but do comment where appropriate.

Second, we look for code which is functionally correct. Our best advice is to read the problem statement carefully, and create a checklist, or even a collection of unit tests by which you can evaluate your solution. If any part of the specification is unclear, please ask for clarification.

Third, we look at the performance of the algorithm. Your solution should perform well on a graph of 100 nodes, fully connected (10,000 edges). The best algorithms continue to perform well on graphs with 1000 nodes and 1,000,000 edges, yielding an answer in under a minute.