Board Games

Being a boardgame connoisseur, you are always looking for a group to play games with. Your friend tells you about a game that is happening on the other side of the neighborhood. However, you'll have to sneak over there in order to play. You already know the layout of your neighborhood and which intersections are too dangeous to travel through (your parents or one of their friends live nearby and they might see you). Given the layout of the neighborhood and which intersections are too dangerous, can you find ALL of the safe paths that will get you to the board game night undetected?

Input

The input file contains one test case. Your program will be run multiple times on the various cases. The first line of input contains the number intersections in the neightborhood $n \leq 1000$. It is assumed that you are trying to travel from intersection 0 to intersection $n-1$. The second line contains an integer $i \leq 10^6$ denoting the number of roads between intersections. The next i lines of input each contain two integers s and t denoting roads between two intersections ($0 \leq s, t \leq n-1$). The next line contains a single integer $x \leq 1000$, the number of intersections that are too dangerous to cross without detection. The next x lines contain the integer representation of each intersection that is too dangerous to cross without detection. Note that is possible that some edges will be listed more than once in the input file. You should treat duplicated edges as a single edge.

Output

For each test case, display ALL possible paths through the neighborhood that are guaranteed to be safe. Each path is a sequence of integers with a single hyphen between each intersection. If there is more than one path, you should print them out in lexicrographic order, with the exception that node ids be considered a single character. For example, 0-2-15 would precede 0-13-15. Even though 13 comes before 2 lexicographically, we are considering 13 to be a single character. Because $2 < 13$, we print 0-2-15 before 0-13-15. You can achieve this ordering quite easily by using a DFS, and always searching smaller numbered nodes before larger numbered ones.

Submission

You should submit your code in Java, C++, or Python. You should configure your makefile properly to run your code.

Sample Input

```
4
4
0 1
1 3
0 2
2 3
1
2

5
6
0 1
0 2
0 3
1 4
2 4
3 4
1
2
```

Sample Output

```
0-1-3

0-1-4
0-3-4
```