Patterns to Match

Problem Description

You've been given two lists: the first is a list of patterns, the second is a list of slash-separated paths. Your job is to print, for each path, the pattern which best matches that path. ("Best" is defined more rigorously below, under "Output Format".)

A pattern is a comma-separated sequence of non-empty fields. For a pattern to match a path, every field in the pattern must exactly match the corresponding field in the path. (Corollary: to match, a pattern and a path must contain the same number of fields.) For example: the pattern `x,y` can only match the path `x/y`. Note, however, that leading and trailing slashes in paths should be ignored, thus `x/y` and `/x/y/` are equivalent.

Patterns can also contain a special field consisting of a *single asterisk*, which is a wildcard and can match any string in the path.

For example, the pattern `A,*,B,*,C` consists of five fields: three strings and two wildcards. It will successfully match the paths `A/foo/B/bar/C` and `A/123/B/456/C`, but not `A/B/C`, `A/foo/bar/B/baz/C`, or `foo/B/bar/C`.

Input Format

The first line contains an integer, N, specifying the number of patterns. The following N lines contain one pattern per line. You may assume every pattern is unique. The next line contains a second integer, M, specifying the number of paths. The following M lines contain one path per line. Only ASCII characters will appear in the input.

Output Format

For each path encountered in the input, print the *best-matching pattern*. The best-matching pattern is the one which matches the path using the fewest wildcards.

If there is a tie (that is, if two or more patterns with the same number of wildcards match a path), prefer the pattern whose leftmost wildcard appears in a field further to the right. If multiple patterns' leftmost wildcards appear in the same field position, apply this rule recursively to the remainder of the pattern.

For example: given the patterns `*,*,c` and `*,b,*`, and the path `/a/b/c/`, the best-matching pattern would be `*,b,*`.

If no pattern matches the path, print `NO MATCH`.

Submission Requirements

You should submit a working program, runnable from a command line, that reads from standard input and prints to standard output. In Unix parlance, for example, it should be runnable like this:

```
cat input_file | python your_program.py > output_file
```

Of course, the actual command line may vary depending on the language you choose; your program file need not be executable on its own. However, it **must** read input directly from stdin and print to stdout.

You may write your program in any of the following languages:

- JavaScript (Node.js)
- Python (2.7 or 3.x)
- Ruby (1.9 or 2.x)

Extra Credit

What's the algorithmic complexity of your program? In other words, how does its running time change as the number of patterns or number of paths increases?

Would your program complete quickly even when given hundreds of thousands of patterns and paths? Is there a faster solution?

Hint: although a correct program is sufficient, there is extra credit for an algorithm that's better than quadratic. Some of our test cases are very large. To pass them all, your program will need to be pretty fast!

Example Input

```
6
*,b,*
a,*,*
*,*,c
foo,bar,baz
w,x,*,*
*,x,y,z
5
/w/x/y/z/
a/b/c
foo/
foo/bar/
foo/bar/baz/
```

Example Output

```
*,x,y,z
a,*,*
NO MATCH
NO MATCH
foo,bar,baz
```

Note about the examples

Because this document is in Markdown format, we have to indent every line in a pre-formatted code block by 4 spaces. Therefore, in both the example input and output above, you should pretend those 4 leading spaces aren't there, because they won't be in the actual test input.

Tips

- Code correctness and quality matter more to us than algorithmic wizardry. Is your program easy to understand? Is it clearly organized and documented? Does it correctly handle all the edges cases? Imagine you are writing a library for other developers to use. How would that affect your design?
- Your program's output must precisely match the expected output. Don't print extraneous or superfluous stuff to stdout.
- The example input and output provided above fail to cover a large number of edge cases. To be sure your program is correct, you may want to supplement it with your own test cases.
- Every line in the input ends with a Unix-style newline (`"\n"`). DOS-style CRLFs (`"\r\n"`) are not used.
- Each line in the output should end with a newline character (that includes the final one). As with the input, use Unix-style newlines.