3860 Schottkey 7th Path

With a typical operating system, a filesystem consists of a number of *directories*, in which reside *files*. These files generally have a canonical location, known as the *absolute path* (such as `/usr/games/bin/kobodl`), which can be used to refer to the file no matter where the user is on a system.

Most operating system environments allow you to refer to files in other directories without having to be so explicit about their locations, however. This is often stored in a variable called `PATH`, and is an ordered list of locations (always absolute paths in this problem) to search for a given name. We will call these *search paths*.

In the brand-new `crash` shell, paths are handled somewhat differently. Users still provide an ordered list of locations that they wish to search for files (their search paths); when a particular filename is requested, however, `crash` tries to be even more helpful than usual. The process it follows is as follows:

- If there is an exact match for the filename, it is returned. Exact matches in locations earlier in the list are preferred. (There are no duplicate filenames in a single location.)
- If there are no exact matches, a filename that has a single *extra* character is returned. That character may be at any point in the filename, but the order of the non-extra characters must be identical to the requested filename. As before, matches in locations earlier in the list are preferred; if there are multiple matches in the highest-ranked location, all such matches in that location are returned.
- If there are no exact matches or one-extra-character matches, files that have *two* extra characters are looked for. The same rules of precedence and multiple matches apply as for the one-extra-character case.
- If no files meet the three criteria above, no filenames are returned. Two characters is considered the limit of "permissiveness" for the `crash` shell.

So, for example, given the two files "`bang`" and "`tang`", they are both one character away from the filename "`ang`" and two from "`ag`". (All characters in this problem will be lowercase.) In the sample data below, both "`cat`" and "`rat`" are one character away from "`at`".

Given a complete list of locations and files in those locations on a system, a set of users each with their own ordered lists of search paths, and a set of files that they wish to search for, what filenames would `crash` return?

For the purposes of simplification, all locations will be described by a single alphabetic string, as will filenames and usernames. Real operating system paths often have many components separated by characters such as slashes, but this problem does not. Also note that users may accidentally refer to nonexistent locations in their search paths; these (obviously) contain no files.

Input

All alphabetic strings in the input will have at least one and at most 20 characters, and will contain no special characters such as slashes or spaces; all letters will be lowercase.

Input to this problem will begin with a line containing a single integer N ($1 \leq N \leq 100$) indicating the number of data sets.

Each data set consists of the following components:

- A line containing a single integer F $(1 \le F \le 100)$ indicating the number of files on the system;
- A series of F lines representing the files on the system, in the format '*location filename*', where *location* and *filename* are both alphabetic strings;
- A line containing a single integer U $(1 \le U \le 10)$ indicating the number of users on the system;
- A series of U stanzas representing the users. Each stanza consists of the following components:
  - A line containing a single alphabetic string which is the user's *username*;
  - A line containing a single integer L $(1 \le L \le 10)$ representing the number of locations in the user's search path; and
  - A series of L lines containing a single alphabetic string apiece listing the locations in the user's search path. The first one is the highest priority, the second (if present) is the second-highest priority, and so on.
- A line containing a single integer S $(1 \le S \le 200)$ indicating the number of file searches to run;
- A series of S lines representing the searches, in the format '*username filename*', where *username* is an alphabetic string that matches one of the users defined in the data set, and *filename* is an alphabetic string that represents the requested filename.

Output

For each data set in the input, output the heading '`DATA SET #`*k*' where k is 1 for the first data set, 2 for the second, etc. Then for each of the S searches in the data set (and in the same order as read from the input) do the following:

- Print the line '*username* `REQUESTED` *filename*' where *filename* is the file requested by *username*.
- For each file (if any) that matches this search, print the line '`FOUND` *filename* `IN` *location*' where *filename* is the file that matched the user's request and that was found in *location*. The list of matching files must be sorted in alphabetical order by *filename*.

Sample Input

```
1
4
food oat
food goat
animal rat
animal cat
2
bob
2
food
animal
bill
1
animal
4
bob at
bob cat
```

```
bill goat
bill at
```

Sample Output

```
DATA SET #1
bob REQUESTED at
FOUND oat IN food
bob REQUESTED cat
FOUND cat IN animal
bill REQUESTED goat
bill REQUESTED at
FOUND cat IN animal
FOUND rat IN animal
```