There are *N* users registered on a website *CuteKittens.com*. Each of them has a unique password represented by *pass[1], pass[2], ..., pass[N]*. As this a very lovely site, many people want to access those awesomely cute pics of the kittens. But the adamant admin does not want the site to be available to the general public, so only those people who have passwords can access it.

Yu, being an awesome hacker finds a loophole in the password verification system. A string which is a *concatenation* of one or more passwords, in any order, is also accepted by the password verification system. Any password can appear 0 or more times in that string. Given access to each of the n passwords, and also have a string $loginAttempt_, determinewhetherthisstringbeacceptedbythepasswordverificationsystemofthewebsite. Ifallofthe$
loginAttempt$ string can be created by concatenating password strings, it is accepted.

For example, if there are *3* users with $passwords = [abra, ka, dabra]$, then some of the valid combinations are `abra` $(passwords[1])$, `kaabra` $(passwords[2] + passwords[1])$, `kadabraka` $(passwords[2] + passwords[3] + passwords[2])$, `kadabraabra` $(passwords[2] + passwords[3] + passwords[1])$ and so on. Supplying `abra ka dabra`, concatenated, passes authentication.

Function Description

Complete the *passwordCracker* function in the editor below. It should return the passwords as a single string in the order required for the password to be accepted, each separated by a space. If it is not possible to form the string, return the string `WRONG PASSWORD`.

passwordCracker has the following parameters:
- *passwords*: a list of password strings
- *loginAttempt*: the string to attempt to create

Input Format

The first line contains an integer *t*, the total number of test cases.

Each of the next t sets of three lines is as follows:
- The first line of each test case contains *n*, the number of users with passwords.
- The second line contains *n* space-separated strings, *passwords[i]*, that represent the passwords of each user.
- The third line contains a string, *loginAttempt*, which *Yu* must test for acceptance.

Constraints

- $1 \leq t \leq 10$
- $1 \leq n \leq 10$
- $passwords[i] \neq passwords[j], 1 \leq i < j \leq N$
- $1 \leq |passwords[i]| \leq 10$, where $i \in [1, n]$
- $1 < |loginAttempt| \leq 2000$
- *loginAttempt* and *passwords[i]* contain only lowercase latin characters ('*a*'-'*z*').

Output Format

For each valid string, *Yu* has to print the actual order of passwords, separated by space, whose concatenation results into *loginAttempt*. If there are multiple solutions, print any of them. If *loginAttempt* can't be accepted by the password verification system, then print `WRONG PASSWORD`.

Sample Input 0

```
3
6
because can do must we what
wedowhatwemustbecausewecan
2
hello planet
helloworld
3
ab abcd cd
abcd
```

Sample Output 0

```
we do what we must because we can
WRONG PASSWORD
ab cd
```

Explanation 0

Sample Case #00: `"wedowhatwemustbecausewecan"` is the concatenation of passwords {`"we"`, `"do"`, `"what"`, `"we"`, `"must"`, `"because"`, `"we"`, `"can"`}. That is

```
loginAttempt = pass[5] + pass[3] + pass[6] + pass[5] +  pass[4] + pass[1] + pass[5] + pass[2]
```

Note that any password can repeat any number of times.

Sample Case #01: We can't create string `"helloworld"` using the strings {`"hello"`, `"planet"`}.

Sample Case #02: There are two ways to create *loginAttempt (*`"abcd"`*)*. Both `pass[2] = "abcd"` and `pass[1] + pass[3] = "ab cd"` are valid answers.

Sample Input 1

```
3
4
ozkxyhkcst xvglh hpdnb zfzahm
zfzahm
4
gurwgrb maqz holpkhqx aowypvopu
gurwgrb
10
a aa aaa aaaa aaaaa aaaaaa aaaaaaa aaaaaaaa aaaaaaaaa aaaaaaaaaa
aaaaaaaaaab
```

Sample Output 1

```
zfzahm
gurwgrb
WRONG PASSWORD
```