Problem M. Password Cracker

OS Linux

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 $loginAttempt$, determine whether this string be accepted by the password verification system of the website. If all of the $loginAttempt$ string can be created by concatenating password strings, it is accepted. In this case, return the passwords in the order they must be concatenated, each separated by a single space on one line. If the password attempt will not be accepted, return 'WRONG PWASSWORD'.

Examples
$passwords =$ `['abra', 'ka', 'dabra']`
$loginAttempt =$ `'abrakadabra'`

Concatenate the passwords in index order $0, 1, 2$ to match 'abrakadabra'. Return 'abra ka dabra'.

$passwords =$ `['abra', 'ka', 'dabra']`
$loginAttempt =$ `'kaabra'`

Concatenate the passwords in index order $1, 0$ to match 'kaabra'. Return 'ka abra'.

$passwords =$ `['ab', 'ba']`
$loginAttempt =$ `'aba'`

Concatenate the passwords in index order $0, 1$ to match 'abba', $1, 0$ to match 'baab', $0, 0$ to match 'abab' or $1, 1$ to match $baba'. No combination of 1 or more passwords can be concatenated to match 'aba'. Return 'WRONG PASSWORD'.

Function Description

Complete the *passwordCracker* function in the editor below.

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

Returns
- *string:* 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`.

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'*).

Input	Output
3 6 because can do must we what wedowhatwemustbecausewecan 2 hello planet helloworld 3 ab abcd cd abcd	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] + pa
```

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.

Input	Output
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	zfzahm gurwgrb WRONG PASSWORD