Emacs is a text editor characterized by its extensibility via plugins. Sometimes Emacs is described as "the extensible, customizable, self-documenting, real-time display editor". Nowadays, Emacs is one of the main contenders in the traditional editor wars of Unix culture.

As a seasoned veteran in the competitive programming scene, Emacs is most of the time your editor of choice. You wish you could always use it but this is not possible because of a feature that is currently not working as expected: a very specific text search capability. In particular, you have gathered proof that searching a text against a regular expression pattern takes a considerable amount of time for some special kinds of patterns. As usual, to the extents of your ego, you are certain that you can implement a much better specific purpose algorithm than the general purpose one currently shipping with Emacs.

A *text* is a string made from lowercase letters ('`a`' to '`z`'). A *pattern* is a string made from lowercase letters and the wild-card symbol ('`*`'). The pattern p *matches* a text t iff p can be found in t as a substring, with each symbol '`*`' in p witnessing an arbitrary substring in t.

For example, let $t =$'`heyhelloyou`', $p_1 =$'`hel*`', $p_2 =$'`*o*e`', and $p_3 =$'`e*o`'. Note that p_1 matches t with '`*`' witnessing, for example, the substring '`loyo`' or even the empty string ''. Also, p_3 matches t with, for example, witness '`lloy`'. However, p_2 does not match t.

You have decided to implement a new algorithm for matching a pattern to a text as part of a new Emacs plugin. Can you come up with a very efficient algorithm?

Input

The input consists of several test cases. Each test case begins with an integer number n indicating the number of patterns on a single line ($1 \leq n \leq 50$). The next line contains the text t ($1 \leq |t| \leq 10^5$). The next n lines contain the patterns $p_1, p_2, \ldots, p_n$ ($1 \leq |p_i| \leq 10^5$ for each $1 \leq i \leq n$), each on a single line.

Output

For each test case output n lines, where the i-th line indicates whether p_i matches t or not. If p_i matches t, then output '`yes`'; otherwise, output '`no`'.

Sample Input

```
4
heyhelloyou
hel*
*o*e
e*o
hello
1
hello
x
```

Sample Output

```
yes
no
yes
yes
no
```