E. BHTML+BCSS

time limit per test: 4 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

This problem is about imaginary languages BHTML and BCSS, which slightly resemble HTML and CSS. Read the problem statement carefully as the resemblance is rather slight and the problem uses very simplified analogs.

You are given a BHTML document that resembles HTML but is much simpler. It is recorded as a sequence of opening and closing tags. A tag that looks like "<tagname>" is called an opening tag and a tag that looks like "</tagname>" is called a closing tag. Besides, there are self-closing tags that are written as "<tagname/>" and in this problem they are fully equivalent to "<tagname></tagname>". All tagnames in this problem are strings consisting of lowercase Latin letters with length from 1 to 10 characters. Tagnames of different tags may coincide.

The document tags form a correct bracket sequence, that is, we can obtain an empty sequence from the given one using the following operations:

- remove any self-closing tag "<tagname/>",
- remove a pair of an opening and a closing tag that go consecutively (in this order) and have the same names. In other words, remove substring "<tagname></tagname>".

For example, you may be given such document: "<header> <a/> </header><footer></footer>" but you may not be given documents "<a>", "<a>", "<a>" or "<a>".

Obviously, for any opening tag there is the only matching closing one — each such pair is called an *element*. A self-closing tag also is an element. Let's consider that one element is nested inside another one, if tags of the first element are between tags of the second one. An element is not nested to itself. For instance, in the example above element "b" is nested in "header" and in "p", but it isn't nested in "a" and "footer", also it isn't nested to itself ("b"). Element "header" has three elements nested in it, and "footer" has zero.

We need the BCSS rules to apply styles when displaying elements of the BHTML documents. Each rule is recorded as a subsequence of words "$x_1\ x_2\ \ldots\ x_n$". This rule has effect over all such elements t, which satisfy both conditions from the list:

- there is a sequence of nested elements with tagnames "x_1", "x_2", ..., "x_n" (that is, the second element is nested in the first one, the third element is nested in the second one and so on),
- this sequence ends with element t (i.e. tagname of element t equals "x_n").

For example, element "b" meets the conditions of the rule "a b" if for element "b" exists element "a" in which it is nested. Element "c" meets the conditions of the rule "a b b c", if three elements exist: "a", "b", "b", and in the chain "a"-"b"-"b"-"c" each following element is nested in the previous one.

Given a BHTML document and a set of BCSS rules, write a program that determines the number of elements that meet the conditions of each rule.

Input

The first line of the input contains a BHTML-document. The document has length from 4 to 10^6 characters. The document has a correct structure, doesn't contain spaces or any other unnecessary characters. Tagnames consist of lowercase Latin letters, their lengths are from 1 to 10 characters.

The second line contains an integer m ($1 \le m \le 200$) — the number of queries. Then m lines contain the queries, one per line. Each query is a sequence $x_1, x_2, ..., x_n$, where x_i is the i-th element of the query, and n ($1 \le n \le 200$) is the number of elements in the query. The elements are separated by single spaces. Each query doesn't begin with and doesn't end with a space. Each query element is a sequence of lowercase Latin letters with length from 1 to 10.

Output

Print m lines, the j-th line should contain the number of elements of the document that correspond to the j-th BCSS-rule. If there are no such elements at all, print on the line `0`.

Examples

input

```
<a><b><b></b></b></a><a><b></b><b><v/></b></a><b></b>
4
a
a b b
a b
b a
```

output

```
2
1
4
0
```

input

```
<b><aa/></b><aa><b/><b/></aa>
5
aa b
b
aa
b aa
a
```

output

```
2
3
2
1
0
```