MMM '14 E - English Essay

Time limit: 4.5s **Memory limit:** 256M

Maniacal Midsummer Marathon 2014 by AL, TL, JJ

The school year is finally coming to a close, and you're dying to get out of school. However, there's still one subject that has yet to be fully taken care of - English. Writing essays is the bane of your existence, but you still have a final essay due that's worth 30% of your final grade. Your English mark is currently at a whopping 64%, and all that really matters now is passing (with a 50%). Luckily, your English teacher also happens to be the CompSci teacher, so some strings may be pulled to save your butt.

You calculated that if you get 20% on this essay, your final grade will be $64 \times 0.7 + 2 \times 0.3 = 50.8\%$ - barely enough to pass! Considering that you're a really bright programmer, your teacher pities you quite a bit, and has decided to make a deal with you. He claims that he will give you a bare minimum of 20% on your essay, provided that it is in proper essay format with formal English, proper grammar, spelling, blah blah blah. You really don't want him to eat his words and fail you by accident, so you decided to make him clearly explain what he wants.

So you ask your teacher, "hey mister, what exactly *is* a formal essay format?" In response, he asks you what you've even been doing the entire year in his class, but you do not succumb to his begging of the question. Finally, he gives in to you because he is impressed by how much you care about this assignment. Being a CompSci teacher, he decides to send you the formal definition of an English essay in Backus-Naur form (BNF).

In BNF, a language or article can be described using a list of symbol definitions. Each definition associates a symbol (represented by an identifier in `<>` angle-brackets) to an expression. **An expression is any combination of concatenations and/or unions of symbols, literal values, and expressions.**

For example, a BNF definition for any integer is as follows:

```
       <integer> ::= "0" | [ "-" ] <natural-number>
<natural-number> ::= <non-zero-digit> { <digit> }
<non-zero-digit> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
         <digit> ::= 0 | <non-zero-digit>
```

In the above example: `::=` indicates a definition. Any series of consecutive expressions indicate a concatenation, meaning that all those expressions must be matched in the order given by the definition. `|` denotes a union, indicating that either the expression on the left or the expression on the right side of the sign can be matched. An expression in curly brackets `{}` indicates that it may be matched 0 or more times. An expression in square brackets `[]` indicates that it may be matched 0 or 1 times. Note that quotations marks around literal values are optional. However, literals that are not enclosed in quotations are assumed to not have any leading, trailing, or joining spaces. That means the definition of `<foo>` below:

```
<foo> ::= ab | a "b" | "a" b | "ab" | <a> <b>
<a> ::= "a"
<b> ::= b
```

will strictly accept nothing else except for the string `ab`. For the purposes of this problem, you may assume that literals will only consist of upper and lowercase alphabetical characters from `A` to `Z`, numerical digits from `0` to `9`, spaces, and punctuation. You do not have to worry about quotation marks, or angled/square/curly brackets being a part of string literals. Symbol names will be between 1 and 20 characters in length, and will only contain lowercase letters from `a` to `z`, numerical digits, and hyphens `-`. We shall assume that the special symbol `<EOL>` represents an end-of-line character. For the sake of simplicity in this problem, **comparisons with definitions and texts are not case sensitive**.

For another example of BNF, we can try to define an actual essay. In order to define an essay with proper spelling, your teacher will first have to give you a large lexicon of valid words defined in BNF like the following. Don't worry, these lists will be provided to you in the same BNF definition list so that all symbols are defined.

```
     <noun> ::= cat | mice | ...
  <pronoun> ::= I | me | you | ...
     <verb> ::= am | are | is | eats | ...
<adjective> ::= cool | smelly | smart | ...
   <adverb> ::= quickly | slowly | ...
  <article> ::= the | a | an | ...
```

Following this, we can define the actual structure of an essay. Below, an essay is defined as a series of paragraphs, which themselves are a series of pseudo-English sentences. For illustration purposes, this definition is very simple and clearly does not cover all English sentences. In actuality, your teacher may send you something much more rigorous and extensive.

Definition	Example Matches/Breakdown

`<essay> ::= <paragraph> { <paragraph> }`
`<paragraph> ::= <sentence> "." { " " <sentence> "." } <EOL>`
`<sentence> ::= <noun-phrase> " " <verb-phrase>`
`<noun-phrase> ::= <pronoun> |`
`<noun> |`
`<article> " " <noun>`
`<verb-phrase> ::= <verb> |`
`<verb-phrase> " " <noun-phrase> |`
`<verb-phrase> " " <adverb> |`
`<verb-phrase> " " <adjective>`
	`"I am cool" + "." + " " + "You are cool" + "."`
	`"The cat" + " " + "eats mice quickly"`
	`"I"`
	`"mice"`
	`"The" + " " + "cat"`
	`"eats"`
	`"eats" + " " + "mice"`
	`"eats mice" + " " + "quickly"`
	`"am" + " " + "cool"`

You may have noticed from the above, definitions can be recursive in nature, calling upon themselves. Furthermore, the order in which symbols are defined does not matter, as long as all of the symbols used in definitions are defined somewhere. The support for recursive definitions can sometimes lead to indeterminate definitions like the following:

```
<foo> ::= <foo>
```

or

```
<foo> ::= <bar>
<bar> ::= <foo>
```

In either of these two cases, we shall consider the number of accepted strings by the defined symbols to be 0. Note that if we had defined `<foo> ::= <foo> | "a"`, then the set of all accepted strings would be $\{$ `a` , `aa` , `aaa` $,\ldots\}$, and the definition would no longer be indeterminate.

Now back to your English essay.

Obviously, you would like to do as little work as possible to pass English. So, given the Backus-Naur form that your teacher has sent you, you would like to write a program that determines the length of the shortest possible string accepted by the symbol `<essay>` in the definition.

Input Specification

The input will contain the proper format of the essay, using the Backus-Naur form variant as described above. It is guaranteed that all used symbols, including the symbol `<essay>`, will be defined exactly once in the input. Definitions may span across multiple lines, although the start of a new definition will always be on a separate line than the end of the previous definition. A literal in quotes will never span across multiple lines. Symbols, literals, `::=`s, `|`s, and brackets will be separated by 1 or more spaces. There may be up to $10\,000$ symbols defined, and the size of the input will not exceed 5 MiB. There may be empty lines in the input which you must ignore.

Note that the Backus-Naur form is not particularly standardized. It is mostly used to describe languages rather than to implement them, so different sources will have different variations. Some constructs used in this problem such as the quantifier brackets are part of the extended Backus-Naur form. The input will **not** employ any other constructs from this form (e.g. single quoted strings, comments, grouping with parentheses, etc.) unless it is explicitly mentioned in this problem statement.

Output Specification

The output should consist of a single integer - the length of the shortest string accepted by the symbol `<essay>`. Since the answer can be very big, output it modulo $1\,000\,000\,007$ $(10^9 + 7)$. An end-of-line symbol `<EOL>` counts as 1 character when matched. If the symbol's definition is indeterminate, output `-1`.

Sample Input 1

```
       <integer> ::= "0" | [ "-" ] <natural-number>
<natural-number> ::= <non-zero-digit> { <digit> }
<non-zero-digit> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
         <digit> ::= 0 | <non-zero-digit>
         <essay> ::= <integer>
```

Sample Output 1

```
1
```

Explanation for Sample 1

Here, an essay is just an integer. The set of shortest strings for this input is { `0` , `1` , `2` , `3` , `4` , `5` , `6` , `7` , `8` , `9` }.

Sample Input 2

```
      <essay> ::= <paragraph> { <paragraph> }
  <paragraph> ::= <sentence> "." { " " <sentence> "." } <EOL>
   <sentence> ::= <noun-phrase> " " <verb-phrase>
<noun-phrase> ::= <pronoun> |
                  <noun> |
                  <article> " " <noun>
<verb-phrase> ::= <verb> |
                  <verb-phrase> " " <noun-phrase> |
                  <verb-phrase> " " <adverb> |
                  <verb-phrase> " " <adjective>
       <noun> ::= cat | mice
    <pronoun> ::= I | me | you
       <verb> ::= am | are | is | eats
  <adjective> ::= cool | smelly | smart
     <adverb> ::= quickly | slowly
    <article> ::= the | a | an
```

Sample Output 2

```
6
```

Explanation for Sample 2

One possible shortest string is `I am.` followed by an EOL character, for a total length of 6.