Question - 1

Approximate Matching

Given three strings, *text*, *prefixString* and *suffixString*, find:

- *prefixScore:* the longest substring of *text* matching the end of *prefixString*
- *suffixScore:* the longest substring of *text* matching the beginning of *suffixString*.

Sum the lengths of those two strings to get the *textScore*. The substring of *text* that begins with the matching prefix and ends with matching suffix is the string to remember. If it is the substring with the highest *textScore*, it is the value you are looking for. If there are other substrings with equal *textScore*, return the lexicographically lowest substring.

For example, if *text = "engine"*, *prefixString = "raven"*, and *suffixString = "ginkgo"*:

- *engine* matches *raven* so *prefixScore = 2*
- *engine* matches *ginkgo* so *suffixScore = 3*
- *textScore = prefixScore + suffixScore = 2 + 3 = 5*
- The substring of *text* with the highest *textScore* is *engin*.

Function Description

Complete the function *calculateScore* in the editor below. The function must return a string that denotes the non-empty substring of *text* having a maximal *textScore*. If there are multiple such substrings, choose the lexicographically smallest substring.

calculateScore has the following parameter(s):

text: a string
prefixString: a string
suffixString: a string

Constraints

- *text*, *prefixString*, and *suffixString* contain lowercase English alphabetic letters *ascii[a-z]* only.
- $1 \leq |text|, |prefixString|, |suffixString| \leq 50$.
- It is guaranteed that there will always be a substring of *text* that matches at least one of the following:
  - One or more characters at the end of *prefixString*.
  - One or more characters at the beginning of *suffixString*.

▼ Input Format for Custom Testing

Input from stdin will be processed as follows and passed to the function.

The first line contains a string *text*.
The next line contains a string *prefixString*.

Help

/

The last line contains a string *suffixString*.

▼ Sample Case 0

Sample Input 0

```
nothing
bruno
ingenious
```

Sample Output 0

```
nothing
```

Explanation 0

- *no*thing matches bru*no* so *prefixScore = 2*
- *noth*ing matches *ing*enious so *suffixScore = 3*
- *textScore = prefixScore + suffixScore = 2 + 3 = 5*

The substring of *text* with the highest *textScore* begins with the prefix *no* and ends with the suffix *ing*: *nothing*.

▼ Sample Case 1

Sample Input 1

```
ab
b
a
```

Sample Output 1

```
a
```

Explanation 1

Given *text = "ab"*, our possible substrings are *sub = "a"*, *sub = "b"*, and *sub = "ab"*.

- *sub = "a"*
  - *prefixString = "b"*: The beginning of *sub* doesn't match the end of *prefixString*, so *prefixScore = 0*.
  - *suffixString = "a"*: The last character of *sub* matches the first character of *suffixString*, so *suffixScore = 1*.
  - *textScore = prefixScore + suffixScore = 0 + 1 = 1*
- *sub = "b"*
  - *prefixString = "b"*: The first character of *sub* matches the last character of *prefixString*, so *prefixScore = 1*.
  - *suffixString = "a"*: The end of *sub* doesn't match the beginning of *suffixString*, so *suffixScore = 0*.
  - *textScore = prefixScore + suffixScore = 1 + 0 = 1*
- *sub = "ab"*
  - *prefixString = "b"*: The beginning of *sub* doesn't match the end of *prefixString*, so *prefixScore = 0*.
  - *suffixString = "a"*: The last character of *sub* doesn't match the first character of *suffixString*, so *suffixScore = 0*.
  - *textScore = prefixScore + suffixScore = 0 + 0 = 0*

Two of these have a *textScore* of *1*, so we return the lexicographically smallest one (i.e., *"a"*).

/

Question - 2
String Patterns

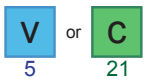
Given the length of a word (*wordLen*) and the maximum number of consecutive vowels that it can contain (*maxVowels*), determine how many unique words can be generated. Words will consist of English alphabetic letters a through z only. Vowels are *v*: {*a, e, i, o, u*}; consonants are *c*: the remaining *21* letters. In the explanations, *v* and *c* represent vowels and consonants.

wordLen = 1

maxVowels = 1

Patterns: *{v, c}*

That means there are *26* possibilities, one for each letter in the alphabet.

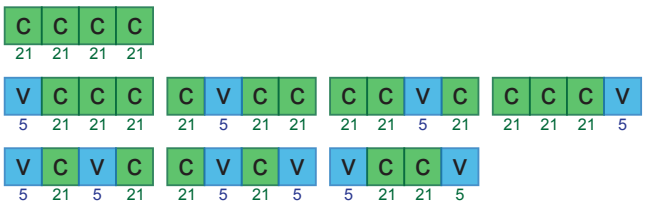


wordLen = 4

maxVowels = 1

Patterns: *{cccc, vccc, cvcc, ccvc, cccv, vcvc, cvcv, vccv}*

There are 412,776 possibilities -- see below:



(21 * 21 * 21 * 21) = 194481

(5 * 21 * 21 * 21) + (21 * 5 * 21 * 21) + (21 * 21 * 5 * 21) + (21 * 21 * 21 * 5) = 4 * 46305 = 185220

(5 * 21 * 5 * 21) + (21 * 5 * 21 * 5) + (5 * 21 * 21 * 5) = 3 * 11025 = 33075

194481 + 185220 + 33075 = 412776 possible solutions.

wordLen = 4

maxVowels = 2

In this case, all of the combinations from the previous example are still valid.

- There are 5 additional patterns to consider, three with 2 vowels (vvcc, cvvc, ccvv) and 2 with 3 vowels (vvcv and vcvv).
- Their counts are 3 * (5 * 5 * 21 * 21) = 3 * 11025 = 33075 and 2 * (5 * 5 * 5 * 21) = 2 * 2625 = 5250.
- The total number of combinations then is 412776 + 33075 + 5250 = 451101.

The result may be a very large number, so return the answer modulo (10^9+7).

/

Note: While the answers will be within the limit of a 32 bit integer, interim values may exceed that limit. Within the function, you may need to use a 64 bit integer type to store them.

Function Description

Complete the function *calculateWays* in the editor below.

calculateWays has the following parameter(s):

int wordLen: the length of a word

int maxVowels: the maximum number of consecutive vowels allowed in a word

Returns

int: the number of well-formed strings that can be created, modulo *1000000007* (10^9+7)

Constraints

- *1 ≤ wordLen ≤ 2500*
- *0 ≤ maxVowels ≤ n*

▼ Input Format Format for Custom Testing

Input from stdin will be processed as follows and passed to the function.

The first line contains an integer *wordLen*, the length of the words to create.
The next line contains an integer *maxVowels*, maximum number of consecutive vowels allowed.

▼ Sample Case 0

Sample Input 0

```
STDIN     Function
-----     --------
2     →   wordLen = 2
1     →   maxVowels = 1
```

Sample Output 0

```
651
```

Explanation 0

Words take the forms *{vc, cv, cc}*. There is a vowel in the first position, the second position or no position. The total number of unique words is *(5 * 21) + (21 * 5) + (21 * 21) = 651* and *651 modulo 1000000007 = 651*.

▼ Sample Case 1

Sample Input 1

```
STDIN     Function
-----     --------
2     →   wordLen = 2
2     →   maxVowels = 2
```

Sample Output 1

/

```
676
```

Explanation 1

Since the words are *2* characters, and there can be *2* consecutive vowels, each position can contain any character. Words take the forms *{vv, vc, cv, cc}*. The total number of unique words is *(26 * 26) = 676* and *676 modulo 1000000007 = 676*.

▼ Sample Case 2

Sample Input 2

```
STDIN       Function
-----       --------
2       →   wordLen = 2
0       →   maxVowels = 0
```

Sample Output 2

```
441
```

Explanation 2

No vowels are allowed in a word, therefore the words are in the form *{cc}*. The total number of unique words is *(21 * 21) = 441* and *441 modulo 1000000007 = 441*.

/