A. Dima and Text Messages

2 seconds, 256 megabytes

Seryozha has a very changeable character. This time he refused to leave the room to Dima and his girlfriend (her hame is Inna, by the way). However, the two lovebirds can always find a way to communicate. Today they are writing text messages to each other.

Dima and Inna are using a secret code in their text messages. When Dima wants to send Inna some sentence, he writes out all words, inserting a heart before each word and after the last word. A heart is a sequence of two characters: the "less" characters (<) and the digit three (3). After applying the code, a test message looks like that: <3$word_1$<3$word_2$<3 ... $word_n$<3.

Encoding doesn't end here. Then Dima inserts a random number of small English characters, digits, signs "more" and "less" into any places of the message.

Inna knows Dima perfectly well, so she knows what phrase Dima is going to send her beforehand. Inna has just got a text message. Help her find out if Dima encoded the message correctly. In other words, find out if a text message could have been received by encoding in the manner that is described above.

Input

The first line contains integer n $(1 \le n \le 10^5)$ — the number of words in Dima's message. Next n lines contain non-empty words, one word per line. The words only consist of small English letters. The total length of all words doesn't exceed 10^5.

The last line contains non-empty text message that Inna has got. The number of characters in the text message doesn't exceed 10^5. A text message can contain only small English letters, digits and signs more and less.

Output

In a single line, print "yes" (without the quotes), if Dima decoded the text message correctly, and "no" (without the quotes) otherwise.

input
3 i love you <3i<3love<23you<3
output
yes

input
7 i am not main in the family <3i<>3am<3the<3<main<3in<3the<3><3family<3
output
no

Please note that Dima got a good old kick in the pants for the second sample from the statement.

B. Lucky Common Subsequence

3 seconds, 512 megabytes

In mathematics, a *subsequence* is a sequence that can be derived from another sequence by deleting some elements without changing the order of the remaining elements. For example, the sequence `BDF` is a subsequence of `ABCDEF`. A *substring* of a string is a continuous subsequence of the string. For example, `BCD` is a substring of `ABCDEF`.

You are given two strings s_1, s_2 and another string called $virus$. Your task is to find the longest common subsequence of s_1 and s_2, such that it doesn't contain $virus$ as a substring.

Input

The input contains three strings in three separate lines: s_1, s_2 and $virus$ ($1 \leq |s_1|, |s_2|, |virus| \leq 100$). Each string consists only of uppercase English letters.

Output

Output the longest common subsequence of s_1 and s_2 without $virus$ as a substring. If there are multiple answers, any of them will be accepted.

If there is no valid common subsequence, output 0.

input
AJKEQSLOBSROFGZ OVGURWZLWVLUXTH OZ
output
ORZ

input
AA A A
output
0

C. Martian Strings

2 seconds, 256 megabytes

During the study of the Martians Petya clearly understood that the Martians are absolutely lazy. They like to sleep and don't like to wake up.

Imagine a Martian who has exactly n eyes located in a row and numbered from the left to the right from 1 to n. When a Martian sleeps, he puts a patch on each eye (so that the Martian morning doesn't wake him up). The inner side of each patch has an uppercase Latin letter. So, when a Martian wakes up and opens all his eyes he sees a string s consisting of uppercase Latin letters. The string's length is n.

"Ding dong!" — the alarm goes off. A Martian has already woken up but he hasn't opened any of his eyes. He feels that today is going to be a hard day, so he wants to open his eyes and see something good. The Martian considers only m Martian words beautiful. Besides, it is hard for him to open all eyes at once so early in the morning. So he opens two non-overlapping segments of consecutive eyes. **More formally, the Martian chooses four numbers a, b, c, d, $(1 \le a \le b < c \le d \le n)$ and opens all eyes with numbers i such that $a \le i \le b$ or $c \le i \le d$.** After the Martian opens the eyes he needs, he reads all the visible characters from the left to the right and thus, he sees some word.

Let's consider all different words the Martian can see in the morning. Your task is to find out how many beautiful words are among them.

Input

The first line contains a non-empty string s consisting of uppercase Latin letters. The strings' length is n $(2 \le n \le 10^5)$. The second line contains an integer m $(1 \le m \le 100)$ — the number of beautiful words. Next m lines contain the beautiful words p_i, consisting of uppercase Latin letters. Their length is from 1 to 1000. All beautiful strings are pairwise different.

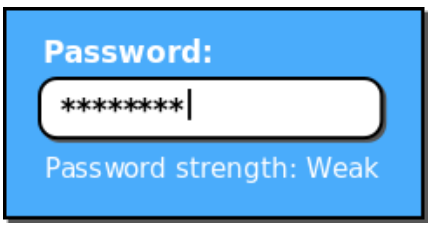
Output

Print the single integer — the number of different beautiful strings the Martian can see this morning.

input
ABCBABA 2 BAAB ABBA
output
1

Let's consider the sample test. There the Martian can get only the second beautiful string if he opens segments of eyes $a = 1, b = 2$ and $c = 4, d = 5$ or of he opens segments of eyes $a = 1, b = 2$ and $c = 6, d = 7$.

D Passwords



It's that time of the year again when you go back to work and need to choose new passwords for all your services. The rules enforced by the system administrators are very strict, and the password you choose must obey the following restrictions:

- It must contain only letters and digits.
- It must have between A and B characters (inclusive).
- It must have at least one lowercase letter, one uppercase letter and one digit.
- It cannot contain any word of a collection of forbidden words (the blacklist).

A word of the blacklist is considered to be contained in the password if it appears as a substring, that is, if it occurs as a consecutive sequence of characters (regardless of it being upper or lower case). For instance, `swerc` is a substring of `SwErC`, `2016swerc2016` or `SWERC16`, but it is not a substring of `ICPC` or `sw16erc`.

Additionally, for the purposes of avoiding the blacklist, you cannot use *l33t*. Specifically, some digits can be interpreted as letters, namely the 0 ('o'), 1 ('i'), 3 ('e'), 5 ('s') and 7 ('t'). This implies that for example `5w3rC` would be an occurrence of `swerc`, and that `abcL337def` contains the word `leet`.

You cannot stop thinking about all these rules and you wonder how many different valid passwords there are... Can you calculate how many passwords would obey these rules?

Task

Given a blacklist with N words and two integers A and B, your task is to compute the number of different valid passwords that exist following the given constraints: made up of only letters and digits; length between A and B (inclusive); at least one lowercase letter, one uppercase letter, and one digit; no blacklisted substring. Since this number can be very big, compute it modulo 1 000 003.

Input

The first line contains two integers, A and B, specifying respectively the minimum and maximum length of the password. The second line contains an integer N, the number of words of the blacklist. The following N lines each contains a string W_i indicating a word in the blacklist. These words are formed only by lowercase letters.

Constraints

$3 \leq$	A	$\leq B \leq$	20	Size of the password
$0 \leq$		$N \leq$	50	Number of words in the blacklist
$1 \leq$	length(W_i)	$\leq$	20	Size of each blacklist word

Output

The output should contain a single line with an integer indicating the number of valid passwords modulo $1\,000\,003$. A valid password is one that respects all of the given constraints.

Sample Input

```
3 5
9
swerc
icpc
fbi
cia
bio
z
hi
no
yes
```

Sample Output

```
607886
```

Sample Explanation

In this case there are exactly $378\,609\,020$ valid passwords and

$$378\,609\,020 \bmod 1\,000\,003 = 607\,886\,.$$

Some examples of valid passwords are: `aA1`, `B23tT`, `1g9K` or `B2j`.

Some examples of invalid passwords are: `aaA` (it does not contain digits), `12a` (it does not contain upper case letters), `a12A34` (length > 5) or `bB10` (contains `bio` as substring).

E. Fake News (hard)

5 seconds, 256 megabytes

Now that you have proposed a fake post for the HC^2 Facebook page, Heidi wants to measure the quality of the post before actually posting it. She recently came across a (possibly fake) article about the impact of fractal structure on multimedia messages and she is now trying to measure the self-similarity of the message, which is defined as

$$\sum_p \mathrm{cnt}(s,p)^2,$$

where the sum is over all nonempty strings p and $\mathrm{cnt}(s,p)$ is the number of occurences of p in s as a **substring**. (Note that the sum is infinite, but it only has a finite number of nonzero summands.)

Heidi refuses to do anything else until she knows how to calculate this self-similarity. Could you please help her? (If you would like to instead convince Heidi that a finite string cannot be a fractal anyway – do not bother, we have already tried.)

Input

The input starts with a line indicating the number of test cases T ($1 \leq T \leq 10$). After that, T test cases follow, each of which consists of one line containing a string s ($1 \leq |s| \leq 100\,000$) composed of lowercase letters (`a-z`).

Output

Output T lines, every line containing one number – the answer to the corresponding test case.

input
4 aa abcd ccc abcc
output
5 10 14 12

A string s contains another string p as a substring if p is a contiguous subsequence of s. For example, `ab` is a substring of `cab` but not of `acb`.

F. Fibonacci Suffix

1 second, 256 megabytes

Let's denote (yet again) the sequence of Fibonacci strings:

$F(0) = \texttt{0}$, $F(1) = \texttt{1}$, $F(i) = F(i-2) + F(i-1)$, where the plus sign denotes the concatenation of two strings.

Let's denote the **lexicographically sorted** sequence of suffixes of string $F(i)$ as $A(F(i))$. For example, $F(4)$ is `01101`, and $A(F(4))$ is the following sequence: `01`, `01101`, `1`, `101`, `1101`. Elements in this sequence are numbered from 1.

Your task is to print m first characters of k-th element of $A(F(n))$. If there are less than m characters in this suffix, then output the whole suffix.

Input

The only line of the input contains three numbers n, k and m ($1 \le n, m \le 200$, $1 \le k \le 10^{18}$) denoting the index of the Fibonacci string you have to consider, the index of the element of $A(F(n))$ and the number of characters you have to output, respectively.

It is guaranteed that k does not exceed the length of $F(n)$.

Output

Output m first characters of k-th element of $A(F(n))$, or the whole element if its length is less than m.

input
4 5 3
output
110

input
4 3 3
output
1

G. Speed Dial

2 seconds, 256 megabytes

Polycarp's phone book contains n phone numbers, each of them is described by s_i — the number itself and m_i — the number of times Polycarp dials it in daily.

Polycarp has just bought a brand new phone with an amazing *speed dial* feature! More precisely, k buttons on it can have a number assigned to it (not necessary from the phone book). To enter some number Polycarp can press one of these k buttons and then finish the number using usual digit buttons (entering a number with only digit buttons is also possible).

Speed dial button can only be used when no digits are entered. No button can have its number reassigned.

What is the minimal total number of **digit number presses** Polycarp can achieve after he assigns numbers to *speed dial* buttons and enters each of the numbers from his phone book the given number of times in an optimal way?

Input

The first line contains two integers n and k ($1 \le n \le 500, 1 \le k \le 10$) — the amount of numbers in Polycarp's phone book and the number of *speed dial* buttons his new phone has.

The i-th of the next n lines contain a string s_i and an integer m_i ($1 \le m_i \le 500$), where s_i is a non-empty string of digits from 0 to 9 inclusive (the i-th number), and m_i is the amount of times it will be dialed, respectively.

It is guaranteed that the total length of all phone numbers will not exceed 500.

Output

Print a single integer — the minimal total number of **digit number presses** Polycarp can achieve after he assigns numbers to *speed dial* buttons and enters each of the numbers from his phone book the given number of times in an optimal way.

input
3 1 0001 5 001 4 01 1
output
14

input
3 1 0001 5 001 6 01 1
output
18

The only *speed dial* button in the first example should have "0001" on it. The total number of digit button presses will be $0 \cdot 5$ for the first number + $3 \cdot 4$ for the second + $2 \cdot 1$ for the third. 14 in total.

The only *speed dial* button in the second example should have "00" on it. The total number of digit button presses will be $2 \cdot 5$ for the first number + $1 \cdot 6$ for the second + $2 \cdot 1$ for the third. 18 in total.