Ador. QuickT9

Most modern mobile phones include T9 technology for typing text messages faster, and your brand new mobile phone is not an exception.

Your mobile phone has the standard keyboard layout:

Button	Letters
2	a,b,c
3	d,e,f
4	g,h,i
5	j,k,l
6	m,n,o
7	p,q,r,s
8	t,u,v
9	w,x,y,z

T9 requires a dictionary of words. At each moment, T9 maintains three strings: D – the current combination of digits, F – the "fixed" part of the message and U – the "unfixed" part of the message. The message displayed on the phone's screen consists of the "fixed" part immediately followed by the "unfixed" part, i.e., it appears as $F + U$. The current combination of digits D is only stored in memory and not displayed. There always is the following correspondence between D and U: they have the same length and the i-th character in U is a letter written on the button with digit equal to the i-th character in D. Additionally, the string U must always be such that there's at least one word in the dictionary that starts with U. For a given combination of digits D, we will call a string U valid if it satisfies the described conditions.

When you start typing a new message, all three strings D, F and U are empty. Then you may do the following:

- press one of the digit buttons (2 – 9): first, the pressed digit is added to the end of D, then U is changed to the lexicographically earliest string that is valid for the new value of D. If there are no such strings, the button press is ignored, so the values of D and U remain the same as before the button press.
- press the Right button $\triangleright$: first, U is appended to the end of F, then both D and U are reset to empty strings.
- press the C button: U is appended to the end of F, then both D and U are reset to empty strings, finally, if F is not empty, the last character is removed from F.
- press the * button: U is changed to the lexicographically next string valid for the current value of D. If there is no such string, it is set to the lexicographically smallest valid string again.

T9 technology is very useful when you need to type a message consisting of dictionary words. However there is a small drawback – typing a word that is not contained in the dictionary becomes much more difficult, so usually you have to type this word part by part (turning T9 off is not considered).

The problem you are facing now is to type a given word using T9. Return the smallest number of button presses needed to type this word on your mobile phone if it is possible at all, otherwise output -1. The word is considered to be typed if F is equal to the word and U is empty.

The dictionary is given in String[] t9. Each element of t9 is a list of words from the dictionary separated by single spaces.

Input

There are multiple test cases in the input file terminated by EOF. For each test case:

First line contains an integer n ($1 \leq n \leq 50$) indicating the number of lists in T9. Each of n lines contains corresponding list of strings. Last line of test case contains a word. Every line has no more than 50 characters. Word contains only lower case letters.

Note if two Strings A and B have the same length, then A comes before B lexicographically if it has an alphabetically earlier character at the first position where the Strings differ. It's possible that the dictionary contains multiple occurrences of the same word.

Output

For each test case, output the desired number in a line.

Sample Input

```
5
aae
bab
abad
bdbd
beta
babe
2
ann
ie
annie
2
ann
amm
annie
4
acac aba aaab
aab aa baa
bba bacade abb
baba
abbaca
3
aaa aab aac
aba abb
ccca
ccc
```

Sample Output

```
9
7
-1
6
10
5
```

(Almost empty page.)

Barr. AppleTrees

Rabbit Hanako likes apples. She decided to plant apple trees along a straight road.

The road has D candidate positions for planting apple trees. These positions are numbered 0 through $D-1$, from left to right. The distance between position x and position y is $|x-y|$ meters ($|x-y|$ denotes the absolute value of $x-y$). She wants to plant N apple trees numbered from 0 to $N-1$ in different positions. The trees may be planted in any order. The i-th tree won't grow if there are other trees which are closer than r_i meters. In other words, if i and j are distinct, the distance between the i-th tree and the j-th tree must be at least $\max(r_i, r_j)$ meters.

Return the number of ways to plant all apple trees modulo 1,000,000,007.

Input

There are multiple test cases in the input file terminated by EOF. For each test case:

First line contains two integers N, D ($1 \leq N \leq 40, 1 \leq D \leq 100000$), next line contains N integers $r_0, r_1, \ldots, r_{N-1}$, ($1 \leq r_i \leq 40$).

Output

For each test case, please output a line contains the desired number.

Sample Input

```
1 10
40
4 4
1 1 1 1
3 4
1 1 2
2 58
5 8
3 47
4 8 9
8 100000
21 37 23 13 32 22 9 39
```

Sample Output

```
10
24
4
2550
28830
923016564
```

(Almost empty page.)

Cudd. NumberMagic

Taro shows a magic trick to Hanako.

Taro: Hello Hanako. I'll show you a magic trick. Please imagine a positive integer less than or equal to 16.

Hanako: OK. I imagined it.

Taro: (Taro shows card 1 to Hanako.) Does this card contain your number?

Hanako: Yes.

Taro: (Taro shows card 2 to Hanako.) Does this card contain your number?

Hanako: No.

Taro: (Taro shows card 3 to Hanako.) Does this card contain your number?

Hanako: Yes.

Taro: (Taro shows card 4 to Hanako.) Does this card contain your number?

Hanako: Yes.

Taro: Your number is 5!

(Card 1 contains 1, 2, 3, 4, 5, 6, 7 and 8. Card 2 contains 1, 2, 3, 4, 9, 10, 11 and 12. Card 3 contains 1, 2, 5, 6, 9, 10, 13 and 14. Card 4 contains 1, 3, 5, 7, 9, 11, 13 and 15.)

Let's generalize this magic trick. Hanako imagines a positive integer less than or equal to N. Each card must contain exactly M different integers between 1 and N, inclusive. Taro must be able to determine Hanako's number correctly using her answers to his questions. Return the minimal number of cards required to perform this magic trick successfully every time.

Input

There are multiple test cases in the input file terminated by EOF. For each test case, there is a line containing two integers N, M, $(2 \le N \le 10^9, 1 \le M \le N - 1)$.

Output

For each test case, please output the desired number.

Sample Input

```
16 8
2 1
3 1
```

Sample Output

```
4
1
2
```

(Almost empty page.)

Dopp. BalancingAct

Dave has a balance with which to weigh objects, and a number of reference weights of known value. All weights are integers between 1 and 100000000 (10^8), inclusive. The balance has 2 pans, each of which can hold any number of objects. The balance will indicate which pan contains more total weight, or that the pans contain equal amounts of weight.

Dave has a problem. His arch nemesis, Earl, has removed the labels from some of the weights, in an attempt to prevent Dave from knowing their values. Dave will attempt to recover the values of the unlabeled weights using the balance and the remaining weights. You are asked to figure out if Dave will succeed.

You will be given the values of all of the weights with their labels intact, and the actual values of the unlabeled weights. output strings with the same number of elements as unlabeled, each "yes" if Dave can determine the value of the corresponding unlabeled weight, or "no" otherwise (quotes for clarity only).

Input

There are multiple test cases in the input file terminated by EOF. For each test case:

First line contains an integer N ($1 \leq N \leq 20$) is the number of labeled weights. Next N integers which are the values of all of the weights with their labels intact. Next integer is M ($1 \leq M \leq 4$) is the number of unlabeled weights. Next M integers which are the actual values of unlabeled weights.

Note that Dave knows that the weights are positive integers not exceeding 100000000 (10^8). See example 2.

Output

For each test case please output the desired results, no empty lines needed.

Sample Input

```
4
9 13 15 16
1
19
1
20
2
10 10
2
33333332 33333334
3
33333333 73 100000000
1
12
4
1 1 2 2
15
31415926 5358979 32384626 43383279 50288419
71693993 75105820 9749445 92307816 40628620
89986280 34825342 11706798 21480865 13282306
4
64709384 46095505 82231725 35940812
```

Sample Output

```
yes
yes
yes
yes
no
yes
no
no
no
no
```

Exot. TicketPrinters

After 2 years of hard work, it is finally time to unveil the ultimate plan to win the lottery. This plan involves mostly cartomancy, falsification and programming. The tarot specialist has already predicted the N numbers of the required lottery tickets, and they are given as an array called $wantedValues$. The falsification specialist has prepared a set of N ticket printers and he has taken a number of security measures to make it very hard to discover the scheme:

- The printers are in separate locations. For convenience, they are all hidden inside houses that are all on a single street. The distances between each consecutive pair of printers are given by the integer array $printerDistance$, which contains $(n-1)$ elements. For each i, the time required to travel between printer i and printer $i+1$ is printerDistance[i] seconds.
- The way a printer functions is by keeping a hidden integer number in its memory. For security, increasing or decreasing the number by one requires 1 second. At any time, it is possible to order the printer to print the number it has in memory. This procedure also takes 1 second. The starting number for printer i is startingValues[i].
- Each printer can print at most one ticket.

Your role as the programmer is to go to each printer and print the required lottery tickets given by $wantedValues$. You are currently in the same location as printer number $currentPrinter$. Since you do not understand why they even hired a programmer for the scheme, you have decided to use programs to save time in this process. The first program you have developed can, given a wanted number, modify the one in the printer by automatically increasing or decreasing it and finally print the wanted number automatically (increasing/decreasing the numbers and printing them still requires one second). You can install this program to each printer once you are its location. The installation doesn't take any time. Once the program is installed, it can work without any administration from your side, so you can immediately proceed to the next printer.

Now your task is to develop the second program that should determine in which order to visit printers and which printer to use for printing of each of the tickets. Return the outcome of the second program, i.e., the minimum time that is necessary to finish printing all the required numbers.

Input

There are multiple test cases in the input file terminated by EOF. For each test case:

First line contains two integers n and $currentPrinter$ ($1 \leq n \leq 16, 0 \leq currentPrinter \leq n-1$). Next line contains $n-1$ integers representing the array of $printerDistance$. Next line contains n integers indicating the array of $startingValues$. Next line contains n integers denoting the array of $wantedValues$. All numbers in array are between 1 and 1000000, and each element in $wantedValues$ will be unique.

Output

For each test case, please output the desired number.

Sample Input

```
4 0
100 20 50
66 78 99 5
99 5 78 66
3 1
50 50
100 200 300
101 201 302
5 2
13 26 39 13
123 12 32 67 1015
1 2 3 4 5
```

Sample Output

```
171
152
1063
```

Frum. StringDecryption

TopCoder Security Agency (TSA) has successfully obtained the string encryption scheme used by the enemy to encrypt strings of digits. This encryption scheme can be described as follows:

Each non-empty maximum set of identical consecutive digits in the original string is replaced by an integer followed by a digit. The integer is the number of consecutive digits that are going to be replaced (without leading zeroes) while the digit is the same digit as those which are going to be replaced. For example, "122" will be replaced by "1122".

The procedure above is applied twice, first to the original string, and then to the result of the first application. For example, "122" becomes "1122" after one application, and becomes "2122" after the next application.

Although encryption is easily implemented with this procedure, retrieving the original data is not since there may be multiple original strings encrypted into the same resulting string. For example, "2122" may be the result of the encryption of the following strings: "122", a string consisting of 112 copies of digit '2' and a string consisting of 22222....22222 (211 '2's) copies of digit '2'.

It is conjectured that the enemy transmits some additional data that allows the encrypted string to be uniquely decrypted. However, TSA has no access to this data, so their only chance to decrypt the messages is to check all possible decryptions.

Given the string S. Compute the number of different strings that will be encrypted to S according to the scheme described above. Return this number modulo 1,000,000,009. If there are no such strings, return 0.

Input

There are multiple test cases in the input file terminated by EOF. For each test case, there is a string S in a line. The string has length no more than 500.

Output

For each test case, please output the desired number in a line.

Sample Input

```
2122
012345
1234056789
1
10000000101

36029876684872327223276091861774662608610223162723
03488815136338720301059585173865765204966825388127
28905156607840356770675251838346615448480878517234
48346801533058114299540857443030369316232988018148
10266938012137248616925283167856138261491565599857
63098906296356837907112034583226442670798371015701
04431120878401385924047425666758653777487585276695
60451685064284613241730873806124686837402534256835
74510620643600499901411494702324867762597665590427
82564618006706585886295436740966342057330943523869

11111111111111111111111111111111111111111111111111
12222222222222211122222222222221112222222222222111
12000000000000211120000000000021112000000000002111
12222220222222211120222222222221112022222222202111
11111120211111111120222222222221112022222222202111
11111120211111111120000000000021112000000000002111
11111120211111111122222222222021112022222222202111
11111120211111111122222222222021112021111111202111
11111120211111111120000000000021112021111111202111
11111122211111111122222222222221112221111111222111


89210
```

Sample Output

```
3
0
1555366
0
1
882582353
371459312
95
```

(Notes: The sample input above is too long such that that numbers are in multiple lines. In the real input file, every input string will be located in a single line.)

Grou. TheBoardingDivOne

John and Brus are going to board a plane.

The boarding can be described using the following simplified model. There are $2 \times n$ cells numbered from 1 to $2 \times n$ from left to right. There are n seats in the plane numbered from 1 to n. The seat i is located near cell $n + i$. There are n passengers numbered from 1 to n. Initially they stand in some order in cells $1, 2, \ldots, n$. The order can be described using a permutation $p_1, p_2, \ldots, p_n$ of integers from 1 to n, where p_i is the number of the passenger who initially stands in cell i. It is known that passenger i wants to take seat i during boarding.

The boarding process can be divided into primitive steps, each of which takes exactly 1 second. During each step, we can check all passengers from right to left and determine for each passenger what he/she will do according to the following rules:

- Denote the number of the passenger we're currently checking as X and the current cell of this passenger as Y.
- If $Y < n+X$ and cell $Y+1$ is currently empty, the passenger will move to this cell. It takes him exactly one step to complete this move, so at the beginning of the next step he/she will be in cell $Y+1$.
- If $Y < n+X$, cell $Y+1$ contains a passenger and the passenger at cell $Y+1$ will perform a move during the current step, the passenger at cell Y will also move to the next cell during the current step (exactly as described in the previous rule).
- If $Y < n+X$, cell $Y+1$ contains a passenger and the passenger at cell $Y+1$ will not move during the current step, the passenger at cell Y will skip the current step (so he/she will still be in cell Y in the beginning of the next step).
- If $Y = n+X$, it means that the passenger in cell Y has reached the cell near which his seat is located. Therefore he will take a seat and it takes 74 steps to do it. So cell Y will be occupied during steps $S, S+1, ..., S+73$ (where S is the number of the current step) because the passenger at this cell will be taking his/her seat. In the beginning of step $S+74$ this cell will become empty because the passenger has completed taking the seat.

The boarding time is defined as the number of steps performed until each passenger has taken his/her seat. Obviously, the boarding time can depend on the initial order of passengers. For example, if $p_1 = 1, p_2 = 2$, the boarding time is 76 (during the first two steps both passengers reach the cells with their seats, and during the next 74 steps they simultaneously take their seats). And if $p_1 = 2, p_2 = 1$, the boarding time is 151 (after one step passenger 1 will reach the cell with his/her seat, during the next 74 steps he/she will take his/her seat and passenger 2 will wait until it's finished, and then passenger 2 will need 76 more steps to reach the required cell and take a seat).

You are given an array `pattern` that imposes some restrictions on the initial order of passengers (described by permutation **p**). The array contains exactly n elements. If `pattern[`i`]` (1-based) is an integer between 1 and n, inclusive, it means that p_i must be equal to `pattern[`i`]`, and if `pattern[`i`]` is -1, it means that p_i can be an arbitrary integer between 1 and n, inclusive.

The initial order of passengers is considered to be good if the boarding time for this order is not greater than $boardingTime$. Find out the number of good initial orders of passengers that match the given pattern.

Input

There are multiple test cases in the input file terminated by EOF. For each test case:

First line contain two integers n and $boardingTime$, $(2 \leq n \leq 18, 2 \leq boardingTime \leq 222)$. Next line contains n integers representing the `pattern` array.

Output

For each test case, please output the desired number.

Sample Input

```
2 100
-1 -1
3 222
-1 2 -1
6 155
2 1 3 5 4 6
7 198
-1 -1 -1 -1 -1 -1 -1
```

Sample Output

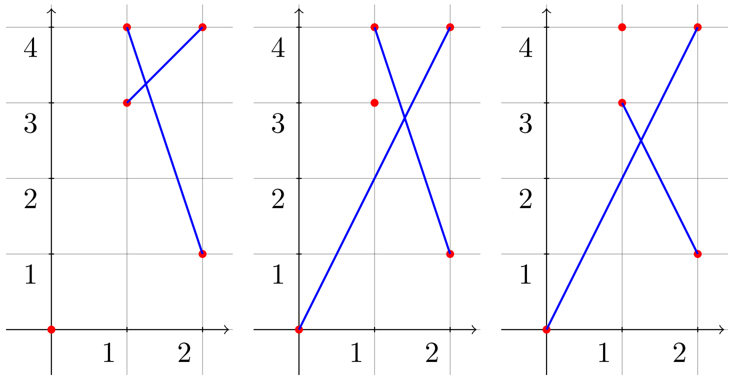
```
1
1
1
233
```

Hanu. LineSegments

The prince has just been introduced to geometry in school. The king wants to check his son's progress by giving him the following geometry problem:

You are given the coordinates of N points on the 2-dimensional plane. No two points are coincident and no three are collinear. Find the number of pairs of intersecting line segments whose endpoints are among the given points. The line segments within each pair must not share any common endpoints.

For example, if the given points are $\{(1,4),(2,1),(0,0),(1,3)$ and $(2,4)\}$, then there are three valid pairs of intersecting line segments as shown in the pictures below.



The king wants you to write a program to solve the above problem so that the answer given by the prince can be checked. The r-th point is (x_r, y_r). The x-coordinates of the points can be generated by using the following recursive definition:

$$\begin{aligned} x_1 &= xFirst, && \text{and} \\ x_i &= (x_{i-1} \times xProd + xAdd) \mod xMod, && \text{for } 2 \le i \le N \end{aligned}$$

The values of y_r are defined similarly using $yFirst, yAdd, yProd$ and $yMod$.

Input

There are multiple test cases in the input file terminated by EOF. For each test case, there are 9 integers $N, xFirst, xAdd, xProd, xMod, yFirst, yAdd, yProd, yMod$ in a line. $2 \le N \le 1200$, $1 \le xMod, yMod \le 1000000$, $0 \le xFirst, xAdd, xProd \le xMod - 1$, $0 \le yFirst, yAdd, yProd \le yMod - 1$. The values of the arguments will be such that among the N generated points (using the above definition), no two points will be coincident and no three will be collinear.

Output

For each test case, please output the desired number.

Sample Input

```
4 4 3 1 5 1 1 1 3
4 1 2 1 3 1 1 1 2
5 1 1 1 3 4 3 2 5
6 1 3 1 5 1 2 1 3
6 215657 553897 915611 930784
193666 323425 130393 654599
```

Sample Output

```
0
1
3
11
15
```