Task 1 : FILTER

The median of 9 numbers is the fifth number when the numbers are arranged in either increasing or decreasing order. For example, the median of the 9 numbers 1, 3 ,4, 1, 2, 6, 8, 4, 10 is 4 because $1 \leq 1 \leq 2 \leq 3 \leq 4 \leq 4 \leq 6 \leq 8 \leq 10$.

An image I is a two dimensional $R \times C$ array of pixels, $3 \leq R \leq 40$, $3 \leq C \leq 40$. A pixel has an integer grey level value V, $0 \leq V \leq 255$.

Median filter is an image processing operation to remove noise. The filter can be implemented by moving a 3×3 window over the image and finding the median of the 9 pixel values covered by the 3×3 window.

For example, given the 6×5 image

$$I = \begin{array}{|c|c|c|c|c|} \hline 49 & 36 & 73 & 62 & 21 \\ \hline 27 & 88 & 14 & 11 & 12 \\ \hline 99 & 18 & 36 & 91 & 21 \\ \hline 45 & 96 & 72 & 12 & 10 \\ \hline 12 & 48 & 49 & 75 & 56 \\ \hline 12 & 15 & 48 & 86 & 78 \\ \hline \end{array}$$

the 4×3 filtered image is

$$J = \begin{array}{|c|c|c|} \hline 36 & 36 & 21 \\ \hline 45 & 36 & 14 \\ \hline 48 & 49 & 49 \\ \hline 48 & 49 & 56 \\ \hline \end{array}$$

You can easily check that as the 3×3 window moves along the top row from left to right, the three window contents are as shown below (the boundaries of the 3×3 window are shown via thick lines)

49	**36**	**73**	62	21
27	**88**	**14**	11	12
99	**18**	**36**	91	21
45	96	72	12	10
12	48	49	75	56
12	15	48	86	78

49	**36**	**73**	**62**	21
27	**88**	**14**	**11**	12
99	**18**	**36**	**91**	21
45	96	72	12	10
12	48	49	75	56
12	15	48	86	78

49	36	**73**	**62**	**21**
27	88	**14**	**11**	**12**
99	18	**36**	**91**	**21**
45	96	72	12	10
12	48	49	75	56
12	15	48	86	78

The corresponding medians for these positions of the 3×3 window are 36, 36 and 21 respectively; hence they constitute the top row of the filtered image J. The pixel values of the other rows of the filtered image J can be found similarly.

Write a program to output an integer which is the number of pixels in the filtered image J whose values are greater than or equal to a threshold T.

For the threshold $T = 40$, the program should output 7 for the above example because there are 7 pixels in the filtered image J whose values are larger than or equal to T.

Input

The input file `FILTER.IN` consists of $R + 2$ lines. The first line contains the two integers R (the number of rows) and C (the number of columns) separated by a blank. The subsequent R lines contain the image: each line contains C pixel values, with a single blank between two adjacent pixel values. The last line contains the single integer T, the threshold value.

Output

The output file `FILTER.OUT` contains a single integer, which is the number of pixels in the filtered image J whose values are larger than or equal to the threshold.

Sample Input 1.

```
3 3
3 4 6
3 1 9
1 9 10
4
```

Sample Output 1.

```
1
```

Sample Input 2.

```
6 5
49 36 73 62 21
27 88 14 11 12
99 18 36 91 21
45 96 72 12 10
12 48 49 75 56
12 15 48 86 78
40
```

Sample Output 2.

```
7
```

Task 2 : BLOCK

You are given a two-dimensional $r \times c$ arrangement of blocks of six different colors. The arrangement consists of up to 100 rows and up to 100 columns. That is, $1 \leq r \leq 100$ and $1 \leq c \leq 100$. The colors are represented by the letters A, B, C, D, E, and F. The blocks always drop as far down as possible until they hit the ground.

Once the blocks sit still, they may eliminate each other as follows. Consider two blocks to be adjacent, if they are either side by side or one on top of the other. Each time the blocks sit still, the area of adjacent blocks of the same color with the biggest number of blocks in it will vanish. If there are several areas with the same biggest number of adjacent blocks, all these areas will vanish.

After they vanish, other blocks may drop down, and a new area of blocks can vanish.

Given a starting arrangement of blocks, you need to find out the number of blocks left when no blocks vanish any more.

Example 1. With the starting arrangement

```
  C
  B A
  B C
C   B
```

the blocks will first drop, to yield the arrangement

```
  C A
  B C
C B B
```

Then, the area containing the color `B` will vanish, leading to the arrangement

```
  C A
    C
C
```

which will make some blocks drop again, yielding the arrangement

```
    A
C C C
```

Finally, the area with color `C` will vanish, and the only remaining block (which has color `A`) will drop. Therefore, the answer is 1.

Example 2. In this example, we start with the arrangement

```
C B B D
C C D D
  C   D
B   B B B
```

After the blocks drop, we get

```
      D
C B B D
C C D D
B C B B B
```

Note that now, there are two consecutive areas with the biggest number of blocks, namely the area with color `C` and the area with color `D`. Both will vanish, leading (after the blocks drop) to

```
    B
B B B B B
```

All these blocks vanish, leaving no blocks in the arrangement. Thus the answer is 0.

Input

The input file `BLOCK.IN` consists of $r + 1$ lines where r is the number of rows in the starting arrangement. The first line consists of the two numbers r and c separated by a blank, where c is the number of columns of the arrangement. Each of the following r lines contains c characters representing the colors of the blocks in the respective row and column. Empty slots are indicated by the letter `X`.

For instance, the file for the starting arrangement of Example 1 above looks like this:

```
4 3
XCX
XBA
XBC
CXB
```

Output

The output file `BLOCK.OUT` contains an integer which is the number of remaining blocks in the final arrangement. For instance, the output file for Example 1 above contains the integer:

```
1
```

Task 3 : COIN

There are k different coin denominations, that is, there are k types of coins, $1 \leq k \leq 20$. Each coin denomination i has a monetary value of v_i cents, $1 \leq i \leq k$. Without loss of generality, we assume $v_1 < v_2 < \ldots < v_k$.

For a given amount M, $0 \leq M \leq 500$, we would like to find the number of distinct ways to form the amount of M cents.

You can assume an infinite supply of coins for each denomination.

Example 1. If the $k = 4$ coin denominations are $v_1 = 10$ cents, $v_2 = 20$ cents, $v_3 = 50$ cents, and $v_4 = 100$ cents, there are 4 ways to form the amount of $M = 50$ cents: (50), $(10, 20, 20)$, $(10, 10, 10, 20)$, and $(10, 10, 10, 10, 10)$.

Example 2. If the $k = 3$ coin denominations are $v_1 = 30$ cents, $v_2 = 70$ cents, and $v_3 = 80$ cents, there are 2 ways to form the amount of $M = 200$ cents: $(30, 30, 30, 30, 80)$ and $(30, 30, 70, 70)$.

Input

The input file COIN.IN consists of three lines. The first line contains the integer k. The second line contains the k integer coin denomination values sorted in increasing order, with a blank between two adjacent values. The third line contains the integer M.

Referring to Example 1 above, the input file looks like:

```
4
10 20 50 100
50
```

Output

The output file COIN.OUT contains a single integer which is the number of ways to form the amount of M cents.

Referring to Example 1 above, the output file looks like:

```
4
```

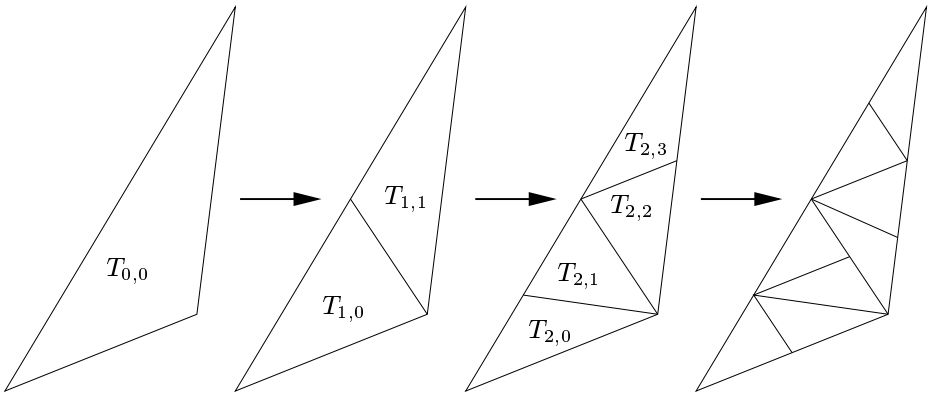
This is a blank page.

This is a blank page.

This is a blank page.

Task 4 : TRIANGLE

Let $T_{0,0}$ be a triangle. Draw an edge between the middle point of the **longest** edge of $T_{0,0}$ and the opposite vertex (if there are two or three edges of maximum length, just choose any of them; this will not change the final result of this problem). This splits $T_{0,0}$ into two smaller triangles $T_{1,0}$ and $T_{1,1}$ (see first two triangles in figure below). We then apply the same process to $T_{1,0}$ and $T_{1,1}$ and obtain four triangles $T_{2,0}, T_{2,1}, T_{2,2}$ and $T_{2,3}$.



When we repeat this process indefinitely for all integer i, it produces an infinite number of triangles $T_{i,j}$ with $j \in \{0, 1, \ldots 2^i - 1\}$. These triangles take only a finite number of different shapes. The goal of this problem is to write a program that counts the number of these different shapes.

For any triangle T, let $l_1(T) \leq l_2(T) \leq l_3(T)$ be the lengths of its three edges. We say that two triangles T and T' are *similar* if and only if it is true that

$$\frac{l_1(T)}{l_1(T')} = \frac{l_2(T)}{l_2(T')} = \frac{l_3(T)}{l_3(T')}.$$

Otherwise, T and T' are not similar. A *similarity class* gathers all the triangles that are similar to a particular triangle T. For instance, equilateral triangles (triangles whose three edges have the same length) form a similarity class.

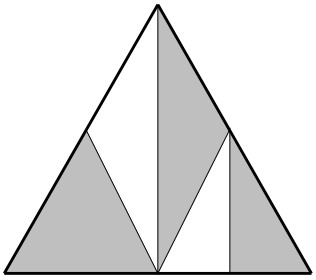
Your program will have to count the number of different similarity classes that appear while subdividing the input triangle $T_{0,0}$. This includes the similarity class of $T_{0,0}$ itself. For instance, if the input triangle $T_{0,0}$ is equilateral, the number of similarity classes that will appear is 3 (see example below).

In order to use only integers, the input triangles $T_{0,0}$ has the following property. For all (i, j), the three numbers $2^i(l_1(T_{i,j}))^2$, $2^i(l_2(T_{i,j}))^2$ and $2^i(l_3(T_{i,j}))^2$ are integers between 1 and 16000. A consequence of this assumption is that the length of the longest edge of $T_{0,0}$ is an even integer (see below).

You may also need the following result. Consider a triangle ABC. Let M be the middle point of BC. Then

$$2AM^2 = AB^2 + AC^2 - \frac{BC^2}{2}.$$

Example. If the input triangle is equilateral, only three similarity classes appear. Any triangle $T_{i,j}$ is similar to one of the three shaded triangles in the picture below.



So in this case, the answer to the problem is 3.

Input

The input file `TRIANGLE.IN` contains three integers, with a blank between two adjacent integers. They represent the lengths of the edges of $T_{0,0}$ in non-decreasing order. For instance, if the input triangle is an equilateral triangle whose edges are of length 12, the input will be

```
12 12 12
```

Output

The output file `TRIANGLE.OUT` contains an integer which is the number of similarity classes that appear while subdividing $T_{0,0}$ indefinitely. This includes the class of $T_{0,0}$. For instance, for the above input file, the output file contains the integer

```
3
```

Task 5 : COMPRESS

In this task, we study the issue of rule-based representation for the compression of very long strings of lower case English letters $a, b, \ldots, z$. Each string is represented by a set of rules. Each rule has two parts, henceforth called left-hand side (lhs) and right-hand side (rhs). The lhs of each rule is a distinct upper case letter, *i.e.* any upper-case letter in the representation appears in the lhs of exactly one rule. The rhs of each rule is a string of upper and lower case letters. An application of a rule is the replacement of its lhs by its rhs. One of the upper case letters is distinguished as the start symbol S. When a string is represented by a set of rules $\mathcal{R}$, then we always get the string from the symbol S by applying the rules of $\mathcal{R}$ repeatedly (until no rule can be applied).

For example, the following two rules form a valid representation of the string *abcabc*

$$\begin{aligned} S &\rightarrow BB \\ B &\rightarrow abc \end{aligned}$$

Starting from S, we can use rule 1 to rewrite S to BB. We then use rule 2 twice to rewrite BB to *abcabc*. Clearly, there can be several rule-based representations of the same string. For example,

$$\begin{aligned} S &\rightarrow aCaC \\ C &\rightarrow bc \end{aligned}$$

is another rule-based representation of the string *abcabc*.

In the interests of compression, we require the rule-based representation of any string to satisfy the following properties.

1. Any pair of adjacent letters (upper or lower case) appearing in the rhs of any rule is unique, *i.e.* it appears only once in *all* the rules.

   This prevents us from representing *abcabc* using the two rules $S \rightarrow XcXc$, $X \rightarrow ab$, because Xc appears twice in the first rule.

2. Except the start symbol (S), every upper-case letter must appear in the right-hand sides of the rules at least twice. This prevents us from representing the string *ab* using the two rules $S \rightarrow aB$, $B \rightarrow b$, since B appears in the rhs only once.

Given an input string, your program should process the string character-by-character from left to right to produce a rule-based compressed representation satisfying the above properties. You should do this in an iterative fashion, *i.e.*

1. read in the next character from the input string;

2. append the character read to the end of the rule containing S (the start symbol);

3. check whether properties 1 and 2 are satisfied in the resultant rules; if not, then repeatedly insert/delete rules to satisfy properties 1 and 2.

For example, the rules representing the string *abcab* are

$$\begin{aligned} S &\rightarrow XcX \\ X &\rightarrow ab \end{aligned}$$

Now, if the next character is c (*i.e.* we consider the string *abcabc*) then we get

$$\begin{aligned} S &\rightarrow XcXc \\ X &\rightarrow ab \end{aligned}$$

This violates property 1, so we insert a new rule as follows

$$\begin{aligned} S &\rightarrow YY \\ Y &\rightarrow Xc \\ X &\rightarrow ab \end{aligned}$$

This now violates property 2 since X appears in the right hand sides of rules only once. So, we delete a rule to get

$$\begin{aligned} S &\rightarrow YY \\ Y &\rightarrow abc \end{aligned}$$

At this stage both properties 1 and 2 are satisfied, and all characters of the string have been read, so we stop.

Input

The input file `COMPRESS.IN` consists of two lines. The first line contains an integer L, $1 \leq L \leq 256$, which is the length of the input string. The second line contains the input string; only lower case English letters $a, \ldots, z$ can appear in the string. The following is a sample input file

```
6
abcabc
```

Output

The output file `COMPRESS.OUT` contains an integer which is the number of rules in the compressed representation of the input string obtained in the described manner. The output for the sample input will be

```
2
```

Task 6 : LABEL

Charles is an auto racing fan and he wants to make his own collection of models. Each model comes with a set of stickers with images of digits. The set of stickers of all models is the same. Using the stickers, Charles labels models with consecutive integers starting from 1. For example, to label the 2070-th model, four stickers are needed: one "2" sticker, two "0" stickers, and one "7" sticker. He can only use stickers that come with the current model or left-over stickers from the previous models. If these stickers do not allow him to label the current model, then he must stop.

Let N be the maximum number of models Charles can label before he runs out of stickers for some digit needed to label a model. It turns out that N can be a very large number even with a small number of stickers that comes with each model. Given the set of stickers, write a program to find the number of digits in the decimal representation of N. You can assume that your calculation involves only decimal numbers of at most 100 digits; that is, decimal numbers $< 10^{100}$.

Example

Suppose the set of stickers has one sticker for each digit "0", "1", $\cdots$, "9". After labelling the 11-th model, the remaining stickers consist of 7 stickers for the digit "1" and 10 stickers for each of the digits "0", "2", "3", $\cdots$, "9".

Charles will run out of stickers for the digit "1" after labelling the 199990-th model. With only one "1" sticker from the next model, he does not have two "1" stickers required for the number 199991 to label the 199991-th model. Therefore the maximum number of models Charles can label is $N = 199990$ and the answer is 6 because there are six digits in N.

Input

The input file `LABEL.IN` consists of one line and it contains ten 1-digit integers, with a blank between two adjacent integers. The first integer is the number of "0" stickers, the second integer is the number of "1" stickers, and so on. The last (the tenth) integer is the number of "9" stickers.

Output

The output file `LABEL.OUT` contains one integer which is the number of digits in the decimal representation of the number of labelled models.

Sample Input 1

```
1 1 1 1 1 1 1 1 1 1
```

Sample Output 1

```
6
```

Sample Input 2

```
3 4 5 4 3 4 5 4 3 4
```

Sample Output 2

```
29
```