Task 1: GRAY

A binary number is a number in base 2, where the individual binary digits, or bits, have weights in powers of two. For example, the decimal number 23 is written as 10111 in binary because:

$$(1\times 2^4)+(0\times 2^3)+(1\times 2^2)+(1\times 2^1)+(1\times 2^0) = (1\times 16)+(1\times 4)+(1\times 2)+(1\times 1) = 23.$$

A Gray code sequence consists of a sequence of binary values, in which each value differs from its immediate predecessor by only a single bit. The following example shows a standard Gray code sequence of 3-bit binary numbers.

Binary sequence	000	001	010	011	100	101	110	111
Standard Gray code sequence	000	001	011	010	110	111	101	100

A very simple algorithm is available to convert a binary number into its equivalent standard Gray code value. For example, to convert binary value 011, we start by copying the first bit:

$$\begin{array}{ccc} 0 & 1 & 1 \\ \downarrow & & \\ 0 & & \end{array}$$

To obtain the second bit, we add the first and second bits of the given binary number, to get the sum $0 + 1 = 1$:

$$\begin{array}{ccccc} \mathbf{0} & + & \mathbf{1} & 1 \\ & & \downarrow & \\ 0 & & 1 & \end{array}$$

To obtain the third bit, we add the second and third bits of the given binary number, to get the sum $1 + 1 = 10$ (in binary), but we discard the carry bit so we just take the rightmost bit 0 of the sum:

$$\begin{array}{cccc} 0 & \mathbf{1} & + & \mathbf{1} \\ & & & \downarrow \\ 0 & 1 & & 0 \end{array}$$

Hence our answer is 010.

In general, except the first bit, the k-th bit of the standard Gray code is obtained by adding the $(k-1)$-th bit and the k-th bit of the given binary number and discarding the carry. This discard-carry addition operation can be described completely as:

$$\begin{array}{r} 0 \\ +\ 0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 0 \\ +\ 1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +\ 0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +\ 1 \\ \hline 0 \end{array}$$

As another example, the 5-bit binary number 10101 is converted to its equivalent standard Gray code value 11111 by applying the above algorithm. The table below shows a few more examples.

Binary value	Equivalent standard Gray code value
01110	01001
111111	100000
1001001	1101101
000111000	000100100

You are to write a program to convert an n-bit binary value into its equivalent n-bit standard Gray code value, where $1 \leq n \leq 20$.

Input

The input file `GRAY.IN` consists of two lines, each line containing one integer.

1. The first line contains the integer n, the number of bits, where $1 \leq n \leq 20$.
2. The second line contains a bit-string of lenght n representing the n-bit binary number.

Output

The output file `GRAY.OUT` contains a bit-string of length n representing the standard Gray code equivalent of the given n-bit binary number.

Input/Output Example

For the first example given above, the input and output files contain:

Sample Input

```
3
011
```

Sample Output

```
010
```

Task 2: LAMP

A building has a long corridor and L ceiling lamps labeled 1, 2, 3, ..., L. Each lamp has an individual switch that can turn the lamp on or off. The building manager hires G security guards (no two guards have the same name), whose job at night is to patrol the corridor and turn the lamps on and off. Each guard is assigned a subset of lamps. During one patrol round, a guard will walk along the corridor, and toggle the lamps assigned to him/her (that is, if a lamp is on, switch it off; if the lamp is off, switch it on). After a guard toggles each lamp assigned to him/her exactly once, the guard will take a break until his/her next patrol round. A guard may have more than one patrol round in one night. The order of patrol for the guards is dictated by a duty roster.

All lamps are off before the guards begin their patrol rounds, and there is only one guard on patrol at any time.

The assignment of lamps to a guard is specified by two positive integers, a_0 and d. The subset of lamps the guard will toggle is

$$\{a_0, a_0 + d, a_0 + 2d, \cdots, a_0 + kd\}$$

where k is the largest integer such that $a_0 + kd \leq L$.

Given the lamp assignment for each guard and the duty roster, find out how many lamps are on after all the guards have finished all of their patrol rounds.

Example

Suppose we have 10 lamps, two security guards (Edi and Lou), and three patrol rounds. Edi's lamp assignment is $(a_0, d) = (1, 4)$ and Lou's lamp assignment is $(a_0, d) = (2, 3)$. The order of patrol is Edi, Lou, Edi.

After the first patrol round by Edi, lamps 1, 5, and 9 will be on. During the second patrol round, Lou toggles lamps 2, 5, and 8. Therefore, after the second patrol round, lamps 1, 2, 8, and 9 are on but lamp 5 has been turned off. On the third patrol round, Edi patrols again, and toggles lamps 1, 5, and 9. Consequently, after all the patrol rounds specified in the duty roster, the lamps that are on are 2, 5, and 8.

The number of lamps that are still on after the guards finish their patrol rounds is therefore 3.

Input

The first line in the input file `LAMP.IN` consists of three positive integers, separated with a blank character. The first number $L \leq 1000$ is the number of lamps. The

second number $G \leq 10$ is the number of security guards, and the third number $R \leq 50$ is the total number of patrol rounds. The next G lines contain the names and lamp assignments of the guards. Each of these G lines consists of the name (exactly 3 English letters), a_0 and d, $(a_0 \leq N)$ separated with a blank character. The subsequent R lines specify the duty roster. Each line contains the name of a guard (guaranteed to have lamp assignment). The order of the guards appearing in the duty roster dictates the order of their patrol.

Output

The output file `LAMP.OUT` contains one integer which is the number of lamps that are on after all guards finish all of their patrol rounds.

Input/Output Example

For the above example, the input and output files contain:

Sample Input

```
10 2 3
Edi 1 4
Lou 2 3
Edi
Lou
Edi
```

Sample Output

```
3
```

Task 3: DOMINO

You are given a set of $2n$ dominos, where $n < 1000$ is an integer. The dominos have width 1 and length 2. It is possible to arrange these dominos in such a way that they form a $4 \times n$ rectangle. For instance, in Figure 1, we have arranged 12 dominos to form a 4×6 rectangle.

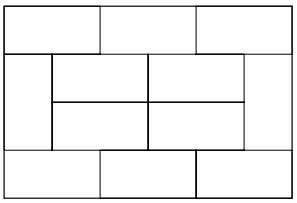


Figure 1: Example with $n = 6$

In fact, when $n > 1$, there are several ways of arranging dominos to form a $4 \times n$ rectangle. For instance, in Figure 2, you can see the 5 different ways of forming a 4×2 rectangle.

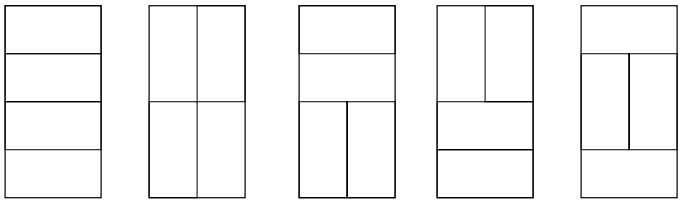


Figure 2: The 5 different ways of forming a 4×2 rectangle.

We denote by R_n the number of different ways of forming a $4 \times n$ rectangle with $2n$ dominos. For instance, $R_2 = 5$, as you can see in Figure 2. As R_n can be quite large, even for small values of n, you are only asked to compute the last three digits of R_n. Your program should output the value of the last three digits of R_n without leading zeros. For instance $R_{17} = 26915305$, so when $n = 17$, your program should output 305 (the last three digits). Another example: $R_2 = 5$ (or 005), so when $n = 2$, your program should output 5. Should the last three digits be zeros for some R_n, your program should simply output 0.

Input

The file `DOMINO.IN` contains the integer n. Remember that $n < 1000$.

Output

You should write an integer giving the value of the last three digits of R_n without leading zeros in the output file `DOMINO.OUT`. (Note that the output value is simply the value R_n mod 1000; that is, the remainder of R_n divided by 1000.)

Input/Output Examples

For the example $R_2 = 5$, the input and output files contain

Sample input 1

```
2
```

Sample output 1

```
5
```

For the example $R_{17} = 26915305$, the input and output files contain

Sample input 2

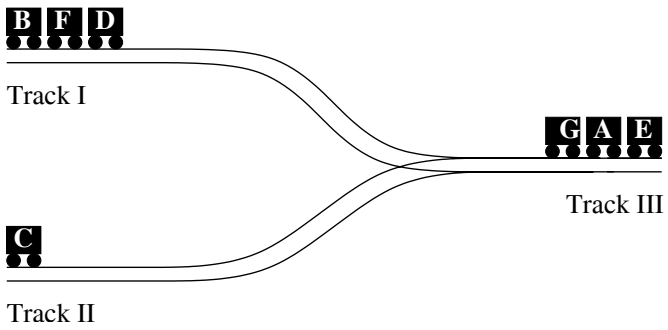
```
17
```

Sample output 2

```
305
```

Task 4: TRAIN

Consider the following configuration of train tracks.

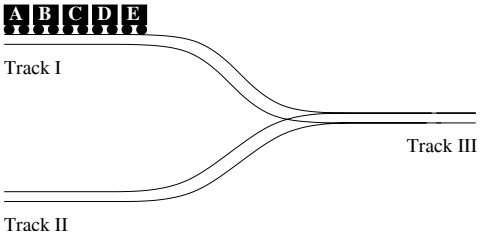


Tracks I, II, and III may hold train cars that can move in one go between Track I and III and between Track II and III, but not between Track I and II. Cars cannot pass each other on a track in one go. Cars can move together in one go. In the situation depicted above, the cars **G** and **A** can move together in one go from Track III to Track I, after which the sequence of cars on Track I is **B**, **F**, **D**, **G**, **A**. However, the car **A** cannot move alone in one go from Track III to Track I. (Because to do so it has to pass car **G** but this is not allowed.) Each track is long enough to hold all cars.

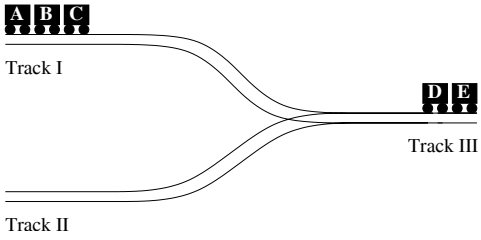
Initially, a sequence of cars is on Track I, and Tracks II and III are empty. The goal is to move cars between Tracks I and III and between Tracks II and III, such that a desired sequence of cars, and no other cars, resides on Track II. The question to be answered by your program is: What is the smallest number of movements that achieves the desired sequence of cars, and no other cars, on Track II? Note that when two or more adjacent cars move together in one go, it counts as one single movement.

Example

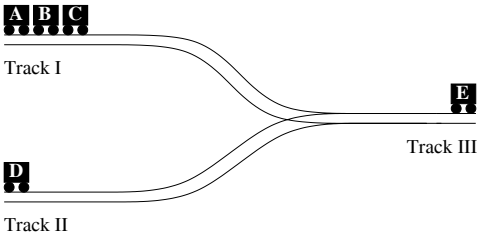
Let us say initially, the cars **A**, **B**, **C**, **D** and **E** are on Track I in the order **ABCDE**. This means that the car **A** is furthest to the left, and car **E** is furthest to the right and closest to Track III. Let us say at the end, we want the sequence **DBC** on Track II. We can achieve this with four movements. First, we move the cars **D** and **E** together from Track I to Track III, then we move car **D** from Track III to Track II, then we move the cars **B** and **C** together from Track I to Track III, and finally, we move the cars **B** and **C** together from Track III to Track II. Figure 3 shows the sequence of movements. The answer therefore is the number 4.



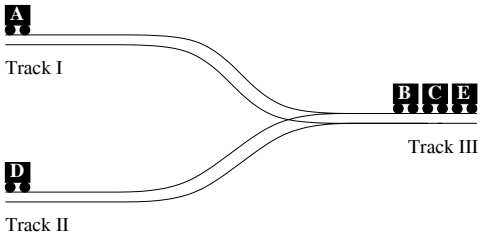
(a) Before first movement



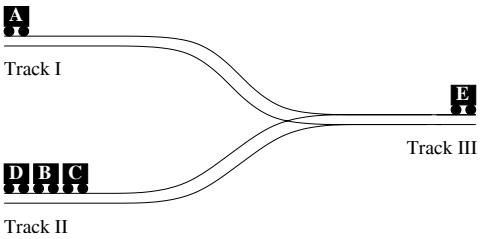
(b) After first movement



(c) After second movement



(d) After third movement



(e) After fourth and final movement

Figure 3: Movements for Example

Input

The input file `TRAIN.IN` consists of three lines. The first line contains two integers, separated with a blank character. The first integer i represents the initial number of cars on Track I, and the second number j represents the number of desired cars on Track II. The second line contains i capital letters, representing the initial sequence of cars on Track I. The third line contains j capital letters representing the desired sequence of cars on Track II.

You may assume that all letters in the second line are distinct, that all letters in the third line are distinct, and that every letter in the third line occurs in the second line, and that $0 < j < 7$, $0 < i < 7$.

Output

The output file `TRAIN.OUT` contains an integer, representing the minimal number of movements to achieve the desired sequence of cars in Track II.

Input/Output Example

For the above example, the input and output files contain:

Sample Input

```
5 3
ABCDE
DBC
```

Sample Output

```
4
```

Task 5: SHIFT

An n-bit shift register is a memory for holding n bits; its contents can be modified in only two possible ways: the contents are left shifted by one position and the rightmost bit is given a 0 or a 1. We call such a modification as a "move". For example, when the contents of a 4-bit shift register are 0001, after one move the contents can become either 0010 (if the rightmost bit is given a 0) or 0011 (if the rightmost bit is given a 1). Consider an n-bit shift register whose contents are initially all zeroes. We need to construct a sequence of moves such that after each move the contents of the shift register are unique, that is, different from all its past contents. Since an n-bit shift register has 2^n possible contents, the longest such sequence can be of length 2^n. It is known that for an n-bit shift register, there are always one or more such sequences of length 2^n. For example when $n = 3$, there are two such sequences of length $2^3 = 8$:

$$A = \quad 000\ 001\ 010\ 101\ 011\ 111\ 110\ 100$$

$$B = \quad 000\ 001\ 011\ 111\ 110\ 101\ 010\ 100$$

LNR (Longest Non-Repeating) Sequences

Given a positive integer n, consider any sequence which satisfies the following properties:

(a) each element of the sequence is a bit-string of length n,

(b) there are 2^n elements in the sequence,

(c) there is no repetition in the sequence,

(d) all the n bits of the first element are zero,

(e) any element in the sequence (apart from the first element) is derived from its previous element via a move described above.

We call any such sequence as a Longest-Non-Repeating or LNR sequence. Given a positive integer n, the set of all LNR sequences (each of which satisfies the above five properties) is denoted as $\Sigma(n)$.

The Smallest LNR Sequence

Any LNR Sequence $\sigma \in \Sigma(n)$ can be associated with a value denoted as $value(\sigma)$ in the following way: we concatenate the 2^n bit-strings in σ to form a single bit-string. The value of this bit-string, considered as a binary number, is $value(\sigma)$. Thus, for the LNR sequence

$$A = 000\ 001\ 010\ 101\ 011\ 111\ 110\ 100$$

in our example above, we have

$$value(A) = 000001010101011111110100$$

We can now define the smallest LNR sequence $\sigma_{small}(n)$ as the LNR sequence σ in $\Sigma(n)$ which yields the smallest $value(\sigma)$.

Example

For $n = 3$, the set of all LNR sequences is given by

$$\Sigma(3) = \{A, B\}$$

where A and B are the sequences discussed earlier. Note that

$$value(A) < value(B),$$

so

$$\sigma_{small}(3) = A.$$

The positions of the bit-strings

$$000,\ 001,\ 010,\ 101,\ 011,\ 111,\ 110,\ 100$$

in A are respectively

$$1, 2, 3, 4, 5, 6, 7, 8.$$

What you need to do

Given a positive integer $n \leq 10$ and bit-string s of length n, your program should compute $\sigma_{small}(n)$ and output the position of s in $\sigma_{small}(n)$. That is, your program should output 1 if s is the first element of $\sigma_{small}(n)$, output 2 if s is the second element of $\sigma_{small}(n)$, etc.

Input

The input file `SHIFT.IN` consists of two lines. The first line contains a positive integer $n \leq 10$. The second line contains a bit-string s of length n.

Output

The output file `SHIFT.OUT` contains a single positive integer giving the position of s in $\sigma_{small}(n)$.

Input/Output Example

In the above example, $\sigma_{small}(3)$ is the sequence A and bit-string 111 is the 6-th element of A. Consequently, we have the following sample input and output.

Sample input

```
3
111
```

Sample Output

```
6
```

Hint

Do not try to construct all LNR sequences and then find the smallest LNR sequence. You can find the smallest LNR sequence by preferring left-shift moves where a "0" is shifted in (over those moves where a "1" is shifted in).

Task 6: RNA

RNA is an important molecule in our body. The structure of an RNA molecule can be described by a pair (S, P) where

1. $S = s_1 s_2 \cdots s_n$ and $P = p_1 p_2 \cdots p_n$ for some positive integer $n \leq 450$.
2. Each element s_i of S is one of the letters A, C, G, U.
3. Each element p_i of P is an asterisk, a left parenthesis, or a right parenthesis.
4. Furthermore, the sequence P is a sequence of balanced parentheses. This means, after ignoring the asterisks (if any), each left parenthesis can be matched by a unique succeeding right parenthesis and each right parenthesis can be matched by a unique preceding left parenthesis.

For example, below shows two RNA structures: $A_1 = (S_1, P_1)$ (on the left) and $A_2 = (S_2, P_2)$ (on the right).

```
ACCCGAACUU          AAUAUCCCGAAU
((**)*(*))          *(**)(**)*()
```

Given two RNA structures $A = (S, P)$ and $A' = (S', P')$, A' is called a substructure of A if there exist integers $i \leq j$ such that $S' = s_i \cdots s_j$ and $P' = p_i \cdots p_j$. (Note that the parentheses of P' should be balanced since $A' = (S', P')$ is an RNA structure.) For example, $A_3 = (S_3, P_3)$ which is

```
CCCG
(**)
```

is a substructure of A_1 but $A_4 = (S_4, P_4)$ which is

```
ACCCGAACU
((**)*(*)
```

is **not** a substructure of A_1 because P_4 is not a sequence of balanced parentheses.

For two RNA structures X and Y, if Z is a substructure of both of them, Z is called a common substructure of X and Y. A longest common substructure is a common substructure with maximum length among all common substructures.

For example, for the above A_1 and A_2, their longest common substructure has length 5 and is

```
CCCGA
(**)*
```

Note that the following example is not a common substructure of A_1 and A_2 because the parentheses are not balanced.

```
CCCGAA
(**)*(
```

In this task, given two RNA structures of length at most 450, you are required to report the length of a longest common substructure.

Input

The input file `RNA.IN` consists of 4 lines. The first two lines are the RNA sequence (first line) and the parenthesis sequence (second line) for the first RNA. The last two lines are the RNA sequence (third line) and the parenthesis sequence (fourth line) for the second RNA. Remember the length of each sequence is at most 450.

Output

The output file `RNA.OUT` contains a single integer which represents the length of a longest common substructure.

Input/Output Example

For the above example, the input and output files contain

Sample Input

```
ACCCGAACUU
((**)*(*))
AAUAUCCCGAAU
*(**)(**)*()
```

Sample Output

```
5
```