Problem A. Coffee Cocktail

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

Sometimes the deadline sneaks up unnoticed and in order to finish a task on time you have to work all night. To stay awake it's easiest to drink coffee.

Programmer Lev from VK usually manages to complete all the assigned tasks in advance but this time he procrastinated the last task for too long and plans to finish it tonight. In order to stay alert he needs at least x caffeine. To achieve this he plans to assemble a set of coffee snacks that are always available in the office.

There are a total of n snacks, each with a mass of m_i and a caffeine content of $k_i\%$. Additionally, each snack has a type (drink, cookie, chocolate bar, etc.) characterized by the integer t_i.

Mixing snacks of different types is not ideal, so Lev wants to use as few different types of snacks as possible. Help him choose a set of snacks with a total caffeine mass of at least x, while minimizing the number of different types of snacks.

Input

The first line of input contains three integers n, q, and x separated by a space, representing the number of different snacks, the number of snack types, and the required caffeine mass, respectively ($1 \le q \le n \le 2 \cdot 10^5$; $1 \le x \le 10^9$).

Each of the following lines contains three integers t_i, m_i, and k_i separated by a space, representing the type, total mass, and percentage caffeine content of the i-th snack ($1 \le t_i \le q$; $1 \le m_i \le 10^9$; $0 \le k_i \le 100$).

Output

Output a single integer representing the minimum number of snack types required to achieve x caffeine. If it is impossible to achieve x caffeine even by taking all the snacks, output "`-1`" (without quotes).

Scoring

Points for each subtask are awarded only if all the tests of that subtask and the required subtasks are successfully passed.

Subtask	Points	Constraints	Required Subtasks
0	–	examples from the statement	
1	12	$q = 1$	
2	16	$k_i = 100$, $t_i \ne t_j$, $m_i = m_j$ for all i, j	
3	18	$t_i \ne t_j$, $k_i = k_j$ for all i, j	2
4	17	$t_i \ne t_j$ for all i, j	2, 3
5	17	$k_i = k_j$ for all i, j	2, 3
6	20	none	0 – 5

Examples

standard input	standard output
4 3 10	2
1 4 25	
2 5 100	
2 4 100	
3 1 10	
5 4 3	1
1 4 50	
1 5 20	
2 2 50	
2 2 25	
4 2 100	

Problem B. Fraction Conversion

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

You are given a decimal fraction, that is, a number containing an integer part and a fractional part, in which both parts are written in the decimal number system. Unfortunately it may have a repeating decimal (fraction period) and working with fractions with repeating decimals is not very convenient.

A fraction with a repeating decimal is written in the format $a.b(c)$, which corresponds to $a.bccc\ldots$, where c is repeated infinitely. For example, 1.25(13) represents the number

$$1 + \frac{2}{10} + \frac{5}{10^2} + \frac{1}{10^3} + \frac{3}{10^4} + \frac{1}{10^5} + \frac{3}{10^6} + \ldots$$

Find the minimum positive base of the number system in which the same number is represented by a non-repeating fraction.

A number is represented as a non-repeating fraction in the number system with base c if it can be represented as a **finite** sum $\sum\limits_i \frac{a_i}{c^i}$. In particular, we consider that for an integer the answer to the problem will be $c = 1$.

Input

The first line of input contains three integers n, m, and k separated by a space — the number of digits in the integer part, non-repeating fractional part, and the repeating part, respectively ($1 \le n \le 12$; $0 \le m, k \le 12$).

The second, third, and fourth lines contain integers a, b, and c — the integer part, fractional part, and period of the number, respectively ($0 \le a, b, c \le 10^{12}$).

Note that the representation of each part can be either an empty string or a number with leading zeros, as the number of occupied positions in the number is important ($1.01 \ne 1.1$).

Output

Output a single integer — the minimum base of the number system in which the given number is represented without a repeating decimal.

Scoring

Points for each subtask are awarded only if all the tests of this subtask and the required subtasks are successfully passed.

Subtask	Points	Constraints	Required Subtasks
0	–	examples from the statement	
1	10	$m = 0$, $k \le 1$	
2	10	$m \le 1$, $k \le 1$	1
3	14	$m \le 6$, $k = 0$	
4	16	$m = 0$, $k \le 6$	1
5	23	$m + k \le 12$	0 – 4
6	27	$m, k \le 12$	0 – 5

Examples

standard input	standard output

`2 1 0`
`10`
`1`
	`10`

`1 2 1`
`0`
`25`
`0`
	`2`

`1 0 1`
`1`
 	
`5`
	`3`

Note

In C++, to read a string until the end, you can use `std::getline`.

Problem C. Public Transportation

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 megabytes

The project of a new city involves the construction of a grid of size $n \times m$ cells. A house with $t_{i,j}$ residents will be located at the intersection of the i-th row and the j-th column.

Three houses in cells (i, j), $(i+a, j)$ and $(i, j+b)$ form a *good triangle* suitable for a public transportation line if:

- $a > 0$ and $b > 0$;
- in the house (i, j), exactly $a + b$ residents live;
- in the house $(i + a, j)$, exactly b residents live;
- and in the house $(i, j + b)$, exactly a residents live.

You can change the construction plan by increasing or decreasing all $t_{i,j}$ by the same integer d. If $t_{i,j}$ should become less than zero, consider it to be equal to zero. Let $T+d$ denote the matrix of values $t_{i,j}+d$, and let $\triangle(T + d)$ be the number of good triangles when the city is constructed accordingly.

Find

$$\sum_{d=-\infty}^{+\infty} \triangle(T + d),$$

or in other words, the total number of good triangles for all possible construction plans.

Input

The first line of input contains two integers n and m separated by a space, representing the number of rows and columns of the grid, respectively ($1 \le n, m \le 1000$).

In the i-th of the following n lines, there are m integers $t_{i,j}$ listed, representing the number of residents in each house of the i-th row of the grid ($1 \le t_{i,j} \le 10^9$).

Output

Output a single integer, the number of sets of three cells that satisfy the given conditions.

Scoring

Points for each subtask are awarded only if all tests of this subtask and the necessary subtasks are successfully passed.

Subtask	Points	Constraints	Required subtasks
0	–	examples from the statement	
1	8	$t_{i,j} \le 2$ for all i, j	
2	13	$n \le 2$	
3	17	$n, m \le 50$, $t_{i,j} \le 50$	0
4	17	$n, m \le 500$	0, 3
5	20	$t_{i,j}$ are generated randomly and uniformly	0
6	25	none	0 – 5

Examples

standard input	standard output

`3 3`
`3 1 1`
`1 2 1`
`1 1 1`
	`2`

`2 4`
`5 4 3 2`
`4 3 2 1`
	`6`

Note

In this problem the input data size is large so it may be useful to use fast IO.

Problem D. Restore Permutation

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 megabytes

This is a run-twice problem. Your solution will be executedn twice.

During the first run you are given p — a permutation of integers from 1 to n. Your program should output a binary string (a string of zeros and ones) of length not exceeding m. The value of m is unknown to your program and depends on the subtask, the corresponding value is indicated in the scoring system table. If your program outputs a string longer than m, it will receive a verdict of "`Wrong Answer`".

Between the runs the jury program will swap two different elements of the original permutation, obtaining permutation q.

During the second run your program will be given permutation q and the binary string output by you in the first run as input. You are required to restore the original permutation.

Input

During the first run, the first input line contains the number 1 and an integer n — the length of the permutation ($2 \le n \le 10^6$). The second line contains n distinct integers p_i separated by a space — the elements of the permutation ($1 \le p_i \le n$).

During the second run, the first and second lines in the same format contain the number 2, the length of the permutation n, and the permutation q, and the third line contains a binary string of zeros (symbol '`0`') and ones (symbol '`1`') of length not exceeding m.

Output

During the first run, output a binary string of length not exceeding m, which will then be used to restore the permutation during the second run.

During the second run, output n distinct integers from 1 to n separated by a space — the elements of the original permutation p.

Interaction Protocol

To avoid receiving incorrect verdicts such as "`Idleness Limit Exceeded`" or "`Security Violation`", terminate the output of each line with a newline character ('`\n`').

Scoring

Points for each subtask are awarded only if all tests of this subtask and the required subtasks are successfully passed.

Subtask	Points	Constraints	Required Subtasks
0	–	examples from the statement, $m = 2000$	
1	7	$n \le 16$, $m = 2000$	0
2	15	$n \le 10^5$, $m = 2 \cdot 10^6$	
3	13	$n \le 2000$, $m = 200$	0, 1
4	11	$m = 32\,000$	2
5	17	$m = 7000$	2, 4
6	10	$m = 2000$	0 – 2, 4, 5
7	19	$m = 500$	0 – 2, 4 – 6
8	18	$m = 200$	0 – 7

Examples

standard input	standard output	Notes
1 4 4 1 2 3	100001010011	Первый запуск

2 4
4 3 2 1
100001010011
	4 1 2 3	Второй запуск

standard input	standard output	Notes
1 5 2 3 4 5 1	1111001001110101000111111100110 0010001001101111000111111111010 011100100	Первый запуск

2 5
2 3 5 4 1
1111001001110101000111111100110
0010001001101111000111111111010
011100100
	2 3 4 5 1	Второй запуск

Note

In the examples long binary strings are displayed with line breaks for correct display in the statement. In reality, there are no such line breaks. Also note that the input and output during the second launch depend on the output during the first launch and may not match the examples shown in the statement.

Problem E. DequeQL

Input file:	`standard input`
Output file:	`standard output`
Time limit:	4 seconds
Memory limit:	256 megabytes

In the modern world there are many different databases based on various structures and principles. In addition to relational (e.g., MySQL) and graph (e.g., GraphQL) databases, a recent intern at VK company proposed the idea of a new database called DequeQL based on deques. Of course not all intern ideas are really good, but being a responsible mentor, Lev decided to implement his idea and test its efficiency.

A *deque* is a data structure that stores a sequence of elements and allows adding a new element or removing an element from either end of the sequence. The basic data block in DequeQL is called a *unit* — an empty deque representing the minimal block of information. The database itself consists of a set of numbered deques nested within each other. A deque that is not nested within any other is called a *root*. Of course, if a unit is not contained within another deque, it is also considered a root.

DequeQL supports four modification operations:

1. `push_back(`d_1`, `d_2`)` — place the root d_1 at the end of the root d_2 (d_1 ceases to be a root);
2. `push_front(`d_1`, `d_2`)` — place the root d_1 at the beginning of the root d_2 (d_1 ceases to be a root);
3. `pop_back(`d`)` — extract the last element from the root d (this element becomes a root);
4. `pop_front(`d`)` — extract the first element from the root d (the extracted element also becomes a root);

Note that operations can only be performed on root elements of the database. Lev has already implemented these operations and decided that DequeQL could be useful in one of the new VK projects if its functionality is expanded to include support for answering the `pop_complexity(`d`)` query: what is the minimum number of `pop_back` or `pop_front` operations needed to make d a root? The structure of the database itself does not change, so there is no need to perform the corresponding `pop` actions.

While Lev is on vacation you have the opportunity to prove yourself as a qualified developer. You are given a database consisting of n units numbered from 1 to n. Implement the required functionality and output the answer to each query.

Input

The first line of input contains two integers n and m — the number of units in the database and the number of queries ($1 \le n, m \le 2 \cdot 10^5$).

The i-th of the following lines contains the i-th query in the format "`<command>` d" or "`<command>` d_1 d_2" where `<command>` is the command or query described in the statement ($1 \le d, d_1, d_2 \le n$). It is guaranteed that database modification operations are only performed on root vertices, and that `pop` operations are only performed on non-empty deques.

Output

For each query, output the answer on a separate line - the minimum number of `pop` operations required to make the corresponding element a root.

Scoring

Points for each subtask are awarded only if all tests of this subtask and the required subtasks are successfully passed.

Subtask	Points	Constraints	Required Subtasks	Testing Feedback
0	–	examples from the statement		full
1	8	$n, m \le 10$	0	full
2	9	each deque directly contains no more than two elements		first error
3	13	each deque directly contains no more than three elements	2	first error
4	12	$n, m \le 2000$	0, 1	first error
5	20	all `pop_complexity` queries come after modification operations		first error
6	14	no `pop` operations		first error
7	24	none	0 – 6	first error

Example

standard input	standard output

`6 11`
`push_back 2 1`
`push_front 3 1`
`pop_back 1`
`push_back 4 2`
`push_front 6 5`
`push_front 2 1`
`push_back 5 1`
`pop_complexity 3`
`pop_complexity 4`
`pop_complexity 5`
`pop_complexity 6`
	`2`
	`2`
	`1`
	`2`