Codeforces Round #117 (Div. 2)

A. Battlefield

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Vasya lagged behind at the University and got to the battlefield. Just joking! He's simply playing some computer game. The field is a flat platform with n trenches dug on it. The trenches are segments on a plane parallel to the coordinate axes. No two trenches intersect.

There is a huge enemy laser far away from Vasya. The laser charges for a seconds, and then shoots continuously for b seconds. Then, it charges for a seconds again. Then it shoots continuously for b seconds again and so on. Vasya knows numbers a and b. He also knows that while the laser is shooting, Vasya must be in the trench, but while the laser is charging, Vasya can safely move around the field. The main thing is to have time to hide in the trench before the shot. If Vasya reaches the trench exactly at the moment when the laser starts shooting, we believe that Vasya managed to hide. Coincidentally, **the length** of any trench in meters numerically **does not exceed** b.

Initially, Vasya is at point A. He needs to get to point B. Vasya moves at speed 1 meter per second in either direction. You can get in or out of the trench at any its point. Getting in or out of the trench takes no time. It is also possible to move in the trench, without leaving it.

What is the minimum time Vasya needs to get from point A to point B, if at the initial time the laser has just started charging? If Vasya cannot get from point A to point B, print -1. If Vasya reaches point B at the moment when the laser begins to shoot, it is believed that Vasya managed to reach point B.

Input

The first line contains two space-separated integers: a and b $(1 \le a, b \le 1000)$, — the duration of charging and the duration of shooting, in seconds.

The second line contains four space-separated integers: A_x, A_y, B_x, B_y $(-10^4 \le A_x, A_y, B_x, B_y \le 10^4)$ — the coordinates of points A and B. It is guaranteed that points A and B do not belong to any trench.

The third line contains a single integer: n $(1 \le n \le 1000)$, — the number of trenches.

Each of the following n lines contains four space-separated integers: x_1, y_1, x_2, y_2 $(-10^4 \le x_i, y_i \le 10^4)$ — the coordinates of ends of the corresponding trench.

All coordinates are given in meters. It is guaranteed that for any trench either $x_1 = x_2$, or $y_1 = y_2$. No two trenches intersect. The length of any trench in meters doesn't exceed b numerically.

Output

If Vasya can get from point A to point B, print the minimum time he will need for it. Otherwise, print number -1.

The answer will be considered correct if the absolute or relative error does not exceed 10^{-4}

Sample test(s)

input
2 4 0 5 6 5 3 0 0 0 4 1 1 4 1 6 0 6 4

output
19.0000000000

input
5 10 0 0 10 10 1 5 0 5 9

output
-1

B. Vasya's Calendar

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Vasya lives in a strange world. The year has n months and the i-th month has a_i days. Vasya got a New Year present — the clock that shows not only the time, but also the date.

The clock's face can display any number from 1 to d. It is guaranteed that $a_i \le d$ for all i from 1 to n. The clock does not keep information about the current month, so when a new day comes, it simply increases the current day number by one. The clock cannot display number $d+1$, so after day number d it shows day 1 (the current day counter resets). The mechanism of the clock allows you to increase the day number by one manually. When you execute this operation, day d is also followed by day 1.

Vasya begins each day checking the day number on the clock. If the day number on the clock does not match the actual day number in the current month, then Vasya manually increases it by one. Vasya is persistent and repeats this operation until the day number on the clock matches the actual number of the current day in the current month.

A year passed and Vasya wonders how many times he manually increased the day number by one, from the first day of the first month to the last day of the n-th month inclusive, considering that on the first day of the first month the clock display showed day 1.

Input

The first line contains the single number d — the maximum number of the day that Vasya's clock can show $(1 \le d \le 10^6)$.

The second line contains a single integer n — the number of months in the year $(1 \le n \le 2000)$.

The third line contains n space-separated integers: a_i $(1 \le a_i \le d)$ — the number of days in each month in the order in which they follow, starting from the first one.

Output

Print a single number — the number of times Vasya manually increased the day number by one throughout the last year.

Sample test(s)

input
4 2 2 2
output
2

input
5 3 3 4 3
output
3

input
31 12 31 28 31 30 31 30 31 31 30 31 30 31
output
7

Note

In the first sample the situation is like this:

- Day 1. Month 1. The clock shows 1. Vasya changes nothing.
- Day 2. Month 1. The clock shows 2. Vasya changes nothing.
- Day 1. Month 2. The clock shows 3. Vasya manually increases the day number by 1. After that the clock shows 4. Vasya increases the day number by 1 manually. After that the clock shows 1.
- Day 2. Month 2. The clock shows 2. Vasya changes nothing.

In total, Vasya manually changed the day number by 1 exactly 2 times.

C. Optimal Sum

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

And here goes another problem on arrays. You are given positive integer len and array a which consists of n integers a_1, a_2, ..., a_n. Let's introduce two characteristics for the given array.

- Let's consider an arbitrary interval of the array with length len, starting in position i. Value $modSum(i, len) = \left| \sum_{j=i}^{i+len-1} a_j \right|$, is the **modular sum** on the chosen interval. In other words, the modular sum is the sum of integers on the chosen interval with length len, taken in its absolute value.
- Value $\max_{1 \le i \le n-len+1} modSum(i, len)$ is the **optimal sum** of the array. In other words, the optimal sum of an array is the maximum of all modular sums on various intervals of array with length len.

Your task is to calculate the optimal sum of the given array a. However, before you do the calculations, you are allowed to produce **no more** than k consecutive operations of the following form with this array: one operation means taking an arbitrary number from array a_i and multiply it by -1. In other words, no more than k times you are allowed to take an arbitrary number a_i from the array and replace it with $-a_i$. Each number of the array is allowed to choose an arbitrary number of times.

Your task is to calculate the maximum possible optimal sum of the array after at most k operations described above are completed.

Input

The first line contains two integers n, len $(1 \le len \le n \le 10^5)$ — the number of elements in the array and the length of the chosen subinterval of the array, correspondingly.

The second line contains a sequence consisting of n integers a_1, a_2, ..., a_n $(|a_i| \le 10^9)$ — the original array.

The third line contains a single integer k $(0 \le k \le n)$ — the maximum allowed number of operations.

All numbers in lines are separated by a single space.

Output

In a single line print the maximum possible optimal sum after no more than k acceptable operations are fulfilled.

Please do not use the `%lld` specifier to read or write 64-bit integers in C++. It is preferred to use `cin`, `cout` streams or the `%I64d` specifier.

Sample test(s)

input
5 3 0 -2 3 -5 1 2
output
10

input
5 2 1 -3 -10 4 1 3
output
14

input
3 3 -2 -5 4 1
output
11

D. Common Divisors

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Vasya has recently learned at school what a number's divisor is and decided to determine a string's divisor. Here is what he came up with.

String a is the divisor of string b if and only if there exists a positive integer x such that if we write out string a consecutively x times, we get string b. For example, string "abab" has two divisors — "ab" and "abab".

Now Vasya wants to write a program that calculates the number of common divisors of two strings. Please help him.

Input

The first input line contains a non-empty string s_1.

The second input line contains a non-empty string s_2.

Lengths of strings s_1 and s_2 are positive and do not exceed 10^5. The strings only consist of lowercase Latin letters.

Output

Print the number of common divisors of strings s_1 and s_2.

Sample test(s)

input
abcdabcd abcdabcdabcdabcd
output
2

input
aaa aa
output
1

Note

In first sample the common divisors are strings "abcd" and "abcdabcd".

In the second sample the common divisor is a single string "a". String "aa" isn't included in the answer as it isn't a divisor of string "aaa".

E. Wooden Fence

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Vasya has recently bought some land and decided to surround it with a wooden fence.

He went to a company called "Wooden board" that produces wooden boards for fences. Vasya read in the catalog of products that the company has at its disposal n different types of wood. The company uses the i-th type of wood to produce a board of this type that is a rectangular a_i by b_i block.

Vasya decided to order boards in this company and build a fence from them. It turned out that the storehouse of the company is so large that Vasya can order arbitrary number of boards of every type. Note that Vasya is allowed to **turn** the boards as he builds the fence. **However, Vasya cannot turn square boards.**

Vasya is required to construct a fence of length l, however, an arbitrary fence won't do. Vasya wants his fence to look beautiful. We'll say that a fence is **beautiful** if and only if the following two conditions are fulfilled:

- **there are no** two successive boards of the same type
- the first board of the fence has an arbitrary length, and the **length** of each subsequent board equals the **width** of the previous one

In other words, the fence is considered beautiful, if the type of the i-th board in the fence is different from the i - 1-th board's type; besides, the i-th board's length is equal to the i - 1-th board's width (for all i, starting from 2).

Now Vasya wonders, how many variants of arranging a fence for his land exist. Your task is to count the number of different beautiful fences of length l.

Two fences will be considered the same if the corresponding sequences of fence boards types and rotations are the same, otherwise the fences are different. Since the sought number can be large enough, you need to calculate the answer modulo 1000000007 $(10^9 + 7)$.

Input

The first line contains two integers n and l $(1 \le n \le 100, 1 \le l \le 3000)$ — the number of different board types and the fence length, correspondingly. Next n lines contain descriptions of board types: the i-th line contains two integers a_i and b_i $(1 \le a_i, b_i \le 100)$ — the sizes of the board of the i-th type. All numbers on the lines are separated by spaces.

Output

Print a single integer — the sought number of variants modulo 1000000007 $(10^9 + 7)$.

Sample test(s)

input
2 3 1 2 2 3
output
2

input
1 2 2 2
output
1

input
6 6 2 1 3 2 2 5 3 3 5 1 2 1
output
20

Note

In the first sample there are exactly two variants of arranging a beautiful fence of length 3:

- As the first fence board use the board of the first type of length 1 and width 2. As the second board use board of the second type of length 2 and width 3.
- Use one board of the second type after you turn it. That makes its length equal 3, and width — 2.