# Problem A. And You All Are The Winners

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 1 second |
| Memory limit: | 1024 mebibytes |

The camp committee is planning a funny event to cheer on the participating teams. The committee provides each team with a pair of integers, $A$ and $B$ ($A \le B$), before the competition, which will be used for the lucky draws after the competition. The committee wants to hold $K$ draws. In each draw, a single integer $C$ is chosen by the committee, and all teams with a pair $(A, B)$ such that $A \le C \le B$ win this draw. To make more teams happy, the committee wants to choose the $K$ integers used in the $K$ draws in advance so that the most teams win. If a team wins multiple draws, it is still counted only once.

For example, imagine five teams are participating in the camp, their pairs are $(1, 2)$, $(1, 4)$, $(3, 6)$, $(4, 7)$, $(5, 6)$, and $K = 2$. If the committee chooses two integers 2 and 4, four teams with $(1, 2)$, $(1, 4)$, $(3, 6)$ and $(4, 7)$ win. The team with $(1, 4)$ wins both draws because their pair contains both chosen integers, but is still counted only once. In fact, all five teams can win if 2 and 5 are chosen. The maximum number of winning teams is five.

Given $n$ pairs of integers for teams and the number of lucky draws $K$, write a program to find the maximum number of winning teams.

## Input

The input starts with a line containing two integers, $n$ and $K$ ($1 \le n \le 10\,000$, $1 \le K \le n$, $1 \le n \cdot K \le 500\,000$), where $n$ is the number of teams and $K$ is the number of lucky draws. Each of the following $n$ lines contains two integers $A$ and $B$ that represent the pair of a team, where $-10^6 \le A \le B \le 10^6$.

## Output

Print exactly one line. The line should contain the maximum number of winning teams. Teams that win more than once should only be counted once.

## Examples

| *standard input* | *standard output* |
|---|---|
| 5 2 | 5 |
| 1 2 | |
| 1 4 | |
| 3 6 | |
| 4 7 | |
| 5 6 | |
| 3 2 | 3 |
| 2 4 | |
| 1 3 | |
| 3 5 | |
| 4 1 | 2 |
| 2 3 | |
| 1 1 | |
| 4 5 | |
| 4 5 | |
| 7 2 | 6 |
| 5 6 | |
| 7 9 | |
| 7 7 | |
| 1 4 | |
| 2 3 | |
| 4 7 | |
| 4 7 | |

## Problem B. Beauty of Integers

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 1 seconds |
| Memory limit: | 1024 mebibytes |

Number theorist Beata is attracted by the beauty of numbers. When we are given an integer $a = \overline{a_1 a_2 \ldots a_n}$ of $n$ digits and a positive integer $k$, $a$ is called $k$-special if the product of all the digits of $a$, that is, $a_1 \cdot a_2 \cdot a_3 \cdot \ldots \cdot a_n$, is divisible by $k$. Note that the number 0 is divisible by every positive integer.

For example, if $a = 2349$ and $k = 12$, then the product of all the digits of $a$, $2 \cdot 3 \cdot 4 \cdot 9 = 216$, is divisible by $k = 12$, so the number 2349 is 12-special. If $a = 2349$ and $k = 16$, then the product of all the digits of $a$, $2 \cdot 3 \cdot 4 \cdot 9 = 216$, is not divisible by $k = 16$, so the number 2349 is not 16-special.

Given three integers $k$, $L$, and $R$, write a program to find $x \bmod 998\,244\,353$ where $x$ is the number of $k$-special numbers among integers in the range $[L, R]$.

## Input

The input has one line containing three integers, $k$, $L$, and $R$ ($1 \le k \le 10^{17}$, $1 \le L \le R \le 10^{20}$).

## Output

Print exactly one line. The line should contain $x \bmod 998\,244\,353$ where $x$ is the number of $k$-special numbers among the numbers in the range $[L, R]$, where both $L$ and $R$ are inclusive in the range.

## Examples

| *standard input* | *standard output* |
|---|---|
| `5 1 20` | `4` |
| `5 50 100` | `19` |
| `15 11 19` | `0` |

## Problem C. Convert Into Irreducible

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 1 second |
| Memory limit: | 1024 mebibytes |

A basic fraction can be represented by three integers $(a\,b\,c)$ which denotes $a + \frac{b}{c}$ where $1 \leq a, b, c \leq 9$. An extended fraction has the form of $(a'\,b'\,c')$ where $a'$, $b'$, and $c'$ are either integers between one and nine or other extended fractions. Note that a basic fraction is also an extended fraction, and the length of the fraction is finite.

Given an extended fraction, we want to express its value as an irreducible fraction.

For example, the irreducible fraction of $((1\,2\,4)\,(5\,2\,3)\,(4\,3\,(2\,7\,3)))$ is as follows.

$$\left(1 + \frac{2}{4}\right) + \frac{5 + \frac{2}{3}}{4 + \frac{3}{2 + \frac{7}{3}}} = \frac{991}{366}$$

Given a string form of an extended fraction, write a program that converts the extended fraction into an irreducible fraction.

## Input

The input starts with a line containing one integer $n$ $(2 \leq n \leq 100)$ where $n$ is the number of characters in the given extended fraction. The second line contains $n$ space-separated characters which are parentheses and digits between 1 and 9: the extended fraction itself.

## Output

Print exactly one line. If the answer is $x/y$, the line should contain two integers $x$ and $y$ which are relatively prime to each other. Otherwise (for example, when the input is not valid), print `-1`. It is guaranteed that the answers fit into 64-bit integers.

## Examples

| *standard input* | *standard output* |
|---|---|
| `5`<br>`( 1 2 3 )` | `5 3` |
| `8`<br>`( 1 2 ( 3 4 5 )` | `-1` |
| `21`<br>`( ( 1 2 4 ) ( 5 2 3 ) ( 4 3 ( 2 7 3 ) ) )` | `991 366` |

## Problem D. DNA Engineering

| | |
|---|---|
| Input file: | *standard input* |
| Output file: | *standard output* |
| Time limit: | 3 seconds |
| Memory limit: | 1024 mebibytes |

Tandem copy is an operation on a DNA where a consecutive sequence of one or more nucleotides is repeated, and the repetitions are directly adjacent to each other; in other words, the tandem copy operation makes a copy of a consecutive sequence of nucleotides and pastes the copy right after the copied sequence. For example, we can tandem copy `ATC` in `ATCG` and obtain `ATCATCG`. Furthermore, we can make another tandem copy operation on the resulting sequence `ATCATCG` and obtain `ATCATTCG`. The following example illustrates a series of tandem copies from `ATCG`, where the underlined sequence is copied on each step:

$$\underline{\texttt{ATC}}\texttt{G} \Rightarrow \texttt{ATCA}\underline{\texttt{T}}\texttt{CG} \Rightarrow \underline{\texttt{AT}}\texttt{CATTCG} \Rightarrow \texttt{ATATCATTCG} \Rightarrow \cdots$$

We say that `ATCG` *produces* all these sequences by tandem copy. It is easy to see that `ATCG` can produce different sequences by selecting a different portion of the sequence for the operation on each step. Furthermore, in principle, `ATCG` can produce infinitely many sequences by making different operations.

Usually, it is more expensive to tandem copy a longer portion of a sequence. For instance,

$$\texttt{ATCG} \Rightarrow \texttt{ATCATCG}$$

is a tandem copy of three nucleotides and thus is more expensive than

$$\texttt{ATCATCG} \Rightarrow \texttt{ATCATTCG}$$

which is a tandem copy of one nucleotide. In other words, the length of the copied portion on each step is crucial to determine the tandem copy cost.

Since it is easy to tandem copy a single nucleotide, it is practical for the genetic engineering lab to store sequences such that every two consecutive nucleotides in a sequence are always different; this helps the lab to reduce the storage space. For instance, since `ATTTG` can be produced by tandem copying `T` twice from `ATG`, it is better for the lab to only store the shorter sequence `ATG` instead of `ATTTG`.

Because of a recent budget cut, the lab can only perform the tandem copy on at most two nucleotides at one time. In other words, the length of the portion copied on each step is at most two. On the other hand, the lab can apply as many tandem copy operations as it desires. For example, given a sequence `ACGT`, we can apply the tandem copy operation on `C` and obtain `ACCGT`, or apply it on the sequence `CG` and obtain `ACGCGT`. But we cannot tandem copy the consecutive sequence `ACG` because its length is more than two.

Given a source string $s$ and a target string $t$, your task is to count the number of valid substrings of $s$. The substring $s'$ is *valid* if one can obtain a string $x$ from $s'$ by applying an appropriate number of tandem copy operations, and $x$ contains $t$ as a substring. Please note that no two consecutive nucleotides in the source string are the same, whereas two consecutive nucleotides in the target string can be the same. For example, `CCA` or `ATTGC` cannot be source strings, but they can be target strings.

Now, given $s$ = `ACATGCAT` and $t$ = `CCACATTT`, we take a substring $s'$ = `CATGC` of $s$ and run a series of tandem copies as follows:

$$s' = \texttt{CATGC} \Rightarrow \texttt{CCATGC} \Rightarrow \texttt{CCACATGC} \Rightarrow \texttt{CCACATTGC} \Rightarrow \texttt{CCACATTTGC}$$

which contains $t$ as its substring.

Here is another substring example. For $s'$ = `CAT`:

$$s' = \texttt{CAT} \Rightarrow \texttt{CACAT} \Rightarrow \texttt{CCACAT} \Rightarrow \texttt{CCACATT} \Rightarrow \texttt{CCACATTT} = t$$

which shows that we can produce the target string from CAT by a series of tandem copies.

It is easy to verify that the total number of valid substrings of $s$ is 14. Note that both the first and the second CAT in $s$ are counted as different valid substrings. Thus, you need to consider all parts of string $s$ as substrings, and count all valid substrings individually.

Here is another example.

When $s =$ AC and $t =$ CA, you can take the substring AC and tandem copy AC. Then, the resulting string is ACAC, which contains CA as its substring. All other substrings of $s$ are unable to produce CA as a substring, and therefore the number of valid substrings is one.

Given a source string $s$ and a target string $t$, where no two consecutive characters in $s$ are the same, write a program that outputs the number of valid substrings $s'$ of $s$.

## Input

The input consists of two lines. The first line is the source string $s$, and the second line is the target string $t$. Each input consists of uppercase English letters, and $1 \le |s|, |t| \le 2 \cdot 10^4$. It is guaranteed that no two consecutive characters in $s$ are the same.

## Output

Print exactly one line. The line should contain the number of valid substrings.

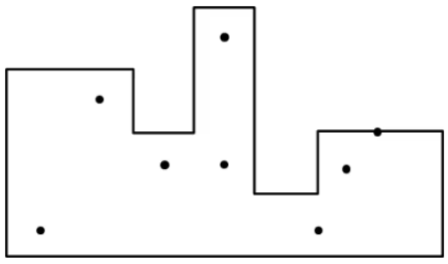
## Examples

| *standard input* | *standard output* |
|---|---|
| ATGTG<br>TTG | 9 |
| CACTGT<br>CCTTG | 6 |
| PQRPQR<br>PQR | 7 |
| BCDBCD<br>BCDBCD | 1 |

## Problem E. Efficient Pest Management

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 1 second |
| Memory limit: | 1024 megabytes |

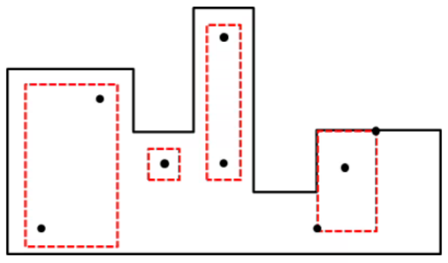
There is a field that borders a straight road. Suppose the road is on the $Ox$ axis. Each boundary edge of the field is either horizontal or vertical. The leftmost and the rightmost edges are vertical and adjacent to the base edge which lies on the road. The length of the base edge is equal to the sum of the lengths of all other horizontal edges. See the figure below:



The dots on the boundary or in the interior of the field represent the locations infested by pests. To efficiently eradicate the infestation, the farmer tries to divide the infested area into several rectangular areas that satisfy the following conditions:

- Each rectangular area must be contained within the field. It is allowed for the edges of a rectangle to overlap the boundary of the field.
- Each edge of a rectangular area is either horizontal or vertical.
- Rectangular areas are completely separated from each other, including their boundaries.
- Each pest infestation location must be contained within one of the rectangular areas. It is allowed for a pest infestation location to lie on an edge of a rectangle.

The figure below shows four rectangular areas covering all pest infestation locations. The farmer wants to minimize the number of rectangular areas for efficient pest management.



Given the boundary of the field and the pest infestation locations, write a program to compute the minimum number of rectangular areas that satisfy the above conditions.

### Input

The input starts with a line containing two integers, $m$ ($4 \le m \le 100\,000$) and $n$ ($0 \le n \le 100\,000$), where $m$ is the number of edges of the field and $n$ is the number of the pest infestation locations.

In the second line, $m$ integers $v_1, v_2, \ldots, v_m$ ($v_1 = v_m = 0$, $0 \le v_i \le 10^6$) are given. They alternate between the $x$ coordinates of the vertical edges and the $y$ coordinates of the horizontal edges. These vertical and horizontal edges are met alternately when traversing the upper boundary of the field clockwise from the left end of the base edge to the right end. So, the first edge in the traversal is a vertical edge from $(v_1, 0)$ to $(v_1, v_2)$, the next edge is a horizontal edge from $(v_1, v_2)$ to $(v_3, v_2)$, and so on.

Starting from the third line, each of the next $n$ lines contains two integers $x$ and $y$ representing the coordinates of a pest infestation location. All locations are on the boundary or in the interior of the field.

It is guaranteed that the length of the base edge is equal to the sum of the lengths of all other horizontal edges.

## Output

Print exactly one line. The line should contain an integer representing the minimum number of rectangular areas that satisfy the above conditions.

## Examples

| standard input | standard output |
|---|---|
| 12 8<br>0 30 20 20 30 40 40 10 50 20 70 0<br>4 5<br>15 26<br>25 15<br>35 15<br>35 35<br>50 5<br>55 15<br>60 20 | 4 |
| 4 0<br>0 10 50 0 | 0 |
| 12 3<br>0 3 2 6 4 1 6 4 8 2 10 0<br>3 5<br>7 3<br>3 1 | 2 |

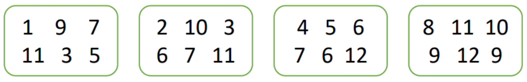
# Problem F. Find The Number

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 1 second |
| Memory limit: | 1024 mebibytes |

David and Harry are friends at the International College of Playing Cards. One day, David met Harry and said:

"I'll do a magic trick for you. Pick any number between 1 and 12. But don't tell me your number! Just keep it in your mind."

Harry secretly chose 11. David then showed Harry the following four cards, one by one. As he was showing each of the cards, he asked: "Is your number on this card?"



So, Harry answered "Yes, yes, no, yes" in this order. After that, David did some magically looking gestures with his arms and legs for a while, then he finally shouted, "I've got your number. It is 11." And Harry was quite surprised because it was exactly the number he chose, and he kept it secret.

David didn't tell Harry the secret of the trick. He only said: "These cards have a great magic power, so they can read your mind and tell me things in a magical language that only I can understand."

How does this work? Can you figure out the secret?

Now, you are to write a program that guesses the numbers in your friends' minds. We can generalize the magic trick as follows: You have $K$ magic cards. There are exactly $M$ integers written on each card. All integers are between 1 and $N$. You perform the magic trick to $F$ friends. Each of them answers your questions truthfully. For each friend, given their answers, guess the number in that friend's mind, or determine that it is impossible.

## Input

The input starts with a line containing four integers, $N$, $K$, $M$, and $F$ ($1 \leq N \leq 500\,000$, $1 \leq K \leq 100$, $1 \leq M \leq 5000$, $1 \leq F \leq 50\,000$).

Each of the following $K$ lines contains $M$ integers between 1 and $N$: the numbers written on the respective magic card.

In each of the following $F$ lines, you are given a string of length $K$ which represents the answers of the respective friend. Each character is either '`Y`' for "yes" or "`N`" for "no". You can assume that each friend has an integer between 1 and $N$ in mind, and everyone answers your questions truthfully.

## Output

Print exactly $F$ lines. For each $i = 1, 2, \ldots, F$, the $i$-th line should contain the number in the $i$-th friend's mind. If it is impossible to uniquely identify some number, print 0 instead.

## Examples

| *standard input* | *standard output* |
|---|---|
| 12 4 6 3<br>1 9 7 11 3 5<br>2 10 3 6 7 11<br>4 5 6 7 6 12<br>8 11 10 9 12 9<br>YYNY<br>NNNY<br>YNNN | 11<br>8<br>1 |
| 13 4 6 4<br>1 9 7 11 3 5<br>2 10 3 6 7 11<br>4 5 6 7 6 12<br>8 11 10 9 12 9<br>YYNY<br>NNNY<br>YNNN<br>NNNN | 11<br>8<br>1<br>13 |
| 14 4 6 4<br>1 9 7 11 3 5<br>2 10 3 6 7 11<br>4 5 6 7 6 12<br>8 11 10 9 12 9<br>YYNY<br>NNNY<br>YNNN<br>NNNN | 11<br>8<br>1<br>0 |

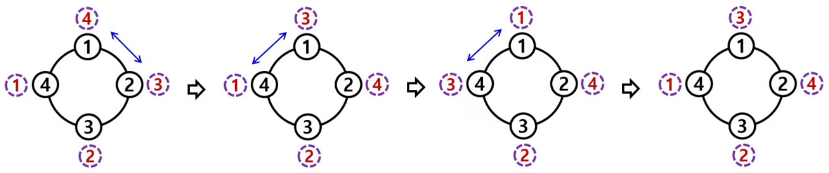
# Problem G. Game with Coins

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 3 seconds |
| Memory limit: | 1024 mebibytes |

A cycle $C$ of length $n$ is a graph with vertices numbered 1 to $n$ where, for each $i = 1, \ldots, n$, the vertices $i$ and $(i \bmod n + 1)$ are connected by an edge.

Consider $n$ coins placed on the cycle, so that each vertex contains one coin. Each coin also has a number from 1 to $n$ on it, but some numbers may be equal. We start in some vertex and walk along the edges, swapping coins in the process. A *swapping walk* $w = (v_1, v_2, \ldots, v_k)$ of length $k \geq 1$ makes $k - 1$ swaps: for $i = 1, 2, \ldots, k - 1$, in this order, we swap the two coins in vertices $v_i$ and $v_{i+1}$. Each two consecutive vertices of the swapping walk must be adjacent in $C$.

The figure below shows the progress of the swapping walk $(2, 1, 4, 1)$ in a cycle with 4 vertices:



You are given two configurations of coins, the initial one and the final one: for each vertex in $C$, which coin is initially in that vertex, and which coin should be there in the end. Find a swapping walk of minimum length which transforms the initial configuration into the final one, or determine that there is no such swapping walk.

For example, in the above figure, the length of the swapping walk $(2, 1, 4, 1)$ is 3. However, the final configuration can also be achieved by the swapping walk $(1, 2)$ with length 1.

## Input

The input starts with a line containing one integer $n$ $(1 \leq n \leq 3000)$, the length of the cycle $C$.

The second line contains $n$ integers between 1 and $n$ (not necessarily distinct): the numbers on the coins initially placed in vertices $1, 2, \ldots, n$.

The third line contains $n$ integers between 1 and $n$ (not necessarily distinct): the numbers on the coins that have to be in vertices $1, 2, \ldots, n$ in the end.

## Output

Print exactly one line. The line should contain the minimum length of a swapping walk that transforms the initial configuration of coins into the final one. If there is no such swapping walk, print $-1$ instead.

## Examples

| `standard input` | `standard output` |
|---|---|
| `4`<br>`4 3 2 1`<br>`3 4 2 1` | `1` |
| `6`<br>`2 1 1 2 2 1`<br>`1 2 2 2 1 1` | `7` |
| `6`<br>`4 1 3 6 2 5`<br>`6 2 1 3 4 5` | `-1` |

## Problem H. HackShuffle

| | |
|---|---|
| Input file: | *standard input* |
| Output file: | *standard output* |
| Time limit: | 1 second |
| Memory limit: | 1024 mebibytes |

The following Python-like pseudo code for function `HackShuffle()` takes a list of positive integers, shuffles the integers in the list in a specific way, and returns the result as a list.

Three specific functions are used below. For a list `L`, the function `len(L)` returns the number of items in `L`. The method `L.append(x)` adds the item `x` to the end of `L`. The method `L.pop(idx)` removes the item at the specified position `idx` from the list `L` (counting from zero) and returns the removed item.

Given a list of positive integers `T`, write a program to reconstruct a list `S` such that `T = HackShuffle(S)`.

```
function HackShuffle( Beta ):
    if len( Beta ) <= 4 :
        exit("Too small Beta")
    Alpha = [] # [] is an empty list
    Gamma = 0
    Delta = len( Beta )

    while( Delta >= 2 ) :
        Omega = Beta[ Gamma ]
        Alpha.append( Omega )
        Beta.pop( Gamma )
        Delta = Delta - 1
        Gamma = ( Omega + Gamma - 1 ) % Delta
    # end of while

    Alpha.append( Beta[ 0 ] )
    Pi = len( Alpha ) - 1
    Omicron = Alpha[ Pi ]
    Lambda = Alpha[ 0 ]
    Rho = Omicron % Pi
    Mu = Alpha[ Rho ]
    Alpha[ 0 ] = Mu
    Alpha[ Rho ] = Lambda

    return ( Alpha )
# end of function HackShuffle
```

### Input

The first line contains an integer $n$ ($5 \le n \le 200\,000$), the length of list `T`.

The following $n$ lines contain integers $T_0, T_1, \ldots, T_{n-1}$, one per line: the list `T` returned from `HackShuffle(S)` ($1 \le T_i \le 100\,000$).

### Output

Print $n$ lines containing integers $S_0, S_1, \ldots, S_{n-1}$, one per line, where `S` is a list such that `T = HackShuffle(S)`.

## Examples

| *standard input* | *standard output* |
|---|---|
| 13<br>113<br>49<br>68<br>91<br>10<br>179<br>2<br>71<br>78<br>45<br>57<br>10<br>88 | 10<br>113<br>179<br>68<br>57<br>45<br>10<br>2<br>88<br>71<br>49<br>78<br>91 |
| 9<br>6<br>8<br>7<br>9<br>5<br>1<br>2<br>4<br>3 | 9<br>8<br>7<br>6<br>5<br>1<br>2<br>3<br>4 |

# Problem I. Integer Matrix

| | |
|---|---|
| Input file: | *standard input* |
| Output file: | *standard output* |
| Time limit: | 1 second |
| Memory limit: | 1024 mebibytes |

We are given a $2 \times n$ matrix $M$ of positive integers. The numbers in each row of $M$ are pairwise distinct. For the $i$-th row $r_i$ of $M$, $i = 1, 2$, we can find the maximum sum $s_i$ of an increasing subsequence contained in $r_i$. For example, with $M$ as in the figure below, $s_1$ is $1+2+3+4+5+6$ and $s_2$ is $2+3+5$. We call $s_1 + s_2$ the maximum sum of increasing subsequences (MSIS).

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 6 & 2 & 3 & 5 & 4 & 1 \end{bmatrix}$$

Once we permute the columns of $M$, its MSIS can change. For example, if we permute the columns of the above matrix $M = [c_1\, c_2\, c_3\, c_4\, c_5\, c_6]$ into $[c_2\, c_3\, c_4\, c_5\, c_6\, c_1]$ as in the figure below, the MSIS becomes 36:

$$\begin{bmatrix} 2 & 3 & 4 & 5 & 6 & 1 \\ 2 & 3 & 5 & 4 & 1 & 6 \end{bmatrix}$$

Given a $2 \times n$ matrix $M$, write a program to output the maximum of MSIS among all possible permutations of the columns of $M$.

## Input

The input starts with a line containing an integer $n$ ($1 \le n \le 10\,000$), the number of columns in the matrix $M$. In the following two lines, the $i$-th line contains $n$ integers of the $i$-th row of $M$. The elements given as input are between 1 and 50 000, and each row does not contain duplicate numbers.

## Output

Print exactly one line. The line should contain the maximum of MSIS among all possible permutations of columns of $M$.

## Examples

| *standard input* | *standard output* |
|---|---|
| 6<br>1 2 3 4 5 6<br>6 2 3 5 4 1 | 36 |
| 5<br>50 40 3 2 1<br>1 2 3 100 200 | 396 |

## Problem J. Jewel Sorting

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 10 seconds |
| Memory limit: | 1024 mebibytes |

Jill has some jewels, and she wants to sort them in non-decreasing order based on size. She uses a unique method to sort the jewels, described below:

Given $n$ jewels, Jill performs a total of $n-1$ steps to sort them. For each step $k$ from 1 to $n-1$:

- She compares the first jewel with the second jewel. If the second jewel is smaller, she swaps their positions.
- She then compares the second jewel with the third jewel. If the third jewel is smaller, she swaps their positions.
- She continues this process until she compares the $(n-k)$-th jewel with the $(n-k+1)$-st jewel, and swaps their positions if the $(n-k+1)$-st jewel is smaller.

Jill's friend Jessie quickly realizes that this is the famous bubble sorting algorithm. To illustrate the inefficiency of this algorithm to Jill, Jessie decides to ask Jill $q$ questions. A question is represented by a tuple $[s, e, m, l, r]$. For a given sequence of $n$ jewels, each question $[s, e, m, l, r]$ asks for the sum of the sizes of jewels from position $l$ to $r$ of the (partially) sorted subsequence after applying the first $m$ steps of Jill's method to the subsequence of jewels from position $s$ to $e$ of the initial sequence.

For instance, consider four ($n = 4$) jewels with sizes $(1, 3, 4, 2)$ and two ($q = 2$) questions: $[2, 4, 1, 2, 2]$ and $[1, 4, 2, 3, 4]$.

For the first question, the subsequence of the sizes from the second ($s = 2$) jewel to the fourth ($e = 4$) jewel is $(3, 4, 2)$. After applying one step ($m = 1$) of Jill's method, it becomes $(3, 2, 4)$. The sum of the sizes of jewels from the second position ($l = 2$) to the second position ($r = 2$) in this (partially) sorted subsequence is 2.

For the second question, the subsequence is $(1, 3, 4, 2)$. After applying two steps, it becomes $(1, 2, 3, 4)$. The sum of the sizes of jewels from position 3 to position 4 in this (partially) sorted sequence is $3 + 4 = 7$.

Given a sequence of $n$ jewels and $q$ questions, write a program that computes the answer for each question.

### Input

The input starts with a line containing two integers, $n$ and $q$ ($2 \le n \le 1\,000\,000$, $1 \le q \le 500\,000$), where $n$ represents the number of jewels and $q$ represents the number of questions.

The second line contains $n$ integers, separated by spaces, representing the sizes of the jewels in their initial order. Each size is between 1 and $10^9$, both inclusive.

Each of the next $q$ lines contains five positive integers $s$, $e$, $m$, $l$, $r$ of query $[s, e, m, l, r]$, separated by spaces, representing a question, where $1 \le s < e \le n$, $1 \le m \le e - s$, and $1 \le l \le r \le e - s + 1$.

### Output

For each of the $q$ questions, output one line with the answer. The answer for a question $[s, e, m, l, r]$ is the sum of the sizes of jewels from position $l$ to $r$ of the partially sorted subsequence after applying the first $m$ steps of Jill's method to the subsequence of jewels from position $s$ to $e$ of the input sequence.

## Examples

| *standard input* | *standard output* |
|---|---|
| 4 2<br>1 3 4 2<br>2 4 1 2 2<br>1 4 2 3 4 | 2<br>7 |
| 5 3<br>4 2 5 1 3<br>1 5 1 3 3<br>1 3 1 3 3<br>2 4 2 1 2 | 1<br>5<br>3 |
| 6 2<br>5 4 5 1 1 4<br>3 6 1 1 3<br>1 6 1 1 4 | 6<br>11 |

# Problem K. King of Luck

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 3 seconds |
| Memory limit: | 1024 mebibytes |

The lottery "King of Luck" awards at most one winner in each round. For each round, $K!$ tickets are produced: each ticket contains $K$ different numbers from 1 to $K$, and no two tickets are identical. Among the tickets produced in each round, $M$ tickets are sold. After each round, the draw is conducted as follows. A total of $N$ distinct numbers ($N \geq K$) are generated randomly, one by one. If, after generation of a number, the relative order of the last $K$ consecutive generated numbers matches the numbers on any of the sold tickets, the draw ends immediately, and the corresponding ticket wins. Note that, since the draw considers only tickets that are sold, some rounds may have no winning ticket.

For instance, consider a round where 6 tickets are produced ($K = 3$). The sequences on the tickets are $(1, 2, 3)$, $(1, 3, 2)$, $(2, 1, 3)$, $(2, 3, 1)$, $(3, 1, 2)$, and $(3, 2, 1)$. Among them, let us say $(1, 2, 3)$ and $(1, 3, 2)$ are sold ($M = 2$). Assume that the following $N = 10$ distinct random numbers are scheduled to be generated: $(20, 35, 10, 7, 99, 53, 72, 33, 88, 16)$. Then the relative order of $(7, 99, 53)$, which is $(1, 3, 2)$, matches the sold ticket $(1, 3, 2)$, so this ticket wins the round.

In another scenario, consider a round where 24 tickets are produced ($K = 4$). The ticket sequences produced are $(1, 2, 3, 4)$, $(1, 2, 4, 3)$, $(1, 3, 2, 4)$, $\ldots$, and $(4, 3, 2, 1)$. Among them, let us say $(1, 2, 3, 4)$, $(1, 2, 4, 3)$, $(3, 4, 1, 2)$, $(4, 1, 2, 3)$, and $(4, 2, 3, 1)$ are sold ($M = 5$). Assume that the following $N = 10$ distinct random numbers are scheduled to be generated: $(19, 31, 9, 1, 89, 48, 63, 30, 78, 12)$. Then the relative order of $(89, 48, 63, 30)$, which is $(4, 2, 3, 1)$, matches the sold ticket $(4, 2, 3, 1)$, so this ticket wins the round.

Given the information about a round of the lottery, including the number of produced tickets, the number sequences of the sold tickets, and the sequence scheduled to be randomly generated for the winning ticket, write a program to find the number sequence of the winning ticket.

## Input

The input starts with a line containing three integers, $K$, $M$, and $N$ ($3 \leq K \leq 10\,000$, $1 \leq M \leq \min(K!, 1000)$, $K \leq N \leq 1\,000\,000$, $3 \leq K \cdot M \leq 100\,000$), where $K$ is the number of numbers on each ticket, $M$ is the number of tickets sold, and $N$ is the length of the randomly generated sequence for the round. Each of the following $M$ lines contains $K$ integers describing a ticket sold in the round. The final line contains $N$ different positive integers $N_i$ ($1 \leq N_i \leq 100\,000\,000$, $1 \leq i \leq N$) which is the number sequence for determining a winner.

## Output

Print exactly one line. The line should contain the number sequence of the winning ticket. If there is no winning ticket, print 0 instead.

## Examples

| *standard input* | *standard output* |
|---|---|
| 3 2 10<br>1 2 3<br>1 3 2<br>20 35 10 7 99 53 72 33 88 16 | 1 3 2 |
| 4 5 10<br>1 2 3 4<br>1 2 4 3<br>3 4 1 2<br>4 1 2 3<br>4 2 3 1<br>19 31 9 1 89 48 63 30 78 12 | 4 2 3 1 |
| 3 3 7<br>1 3 2<br>2 3 1<br>2 1 3<br>11 22 33 44 55 66 77 | 0 |

## Problem L. Logistics of Bubble Gum

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 1 seconds |
| Memory limit: | 1024 mebibytes |

There is only one railway line connecting $(n+1)$ cities developed along the coastline. The cities along the coast are numbered sequentially by integers between 0 and $n$. City $(i-1)$ and city $i$ $(1 \le i \le n)$ are connected by rail, and other pairs of cities are not connected by rail.

Since every city except city 0 is famous as a tourist destination, every city $i$ $(1 \le i \le n)$ excluding city 0 is preparing a variety of goods to welcome travelers ahead of the tourist season. Worldwide famous Bubble Gum is the most popular product in every city. However, the supplier of this product is located in city 0.

There is only one store that sells Bubble Gum in each city $i$ $(1 \le i \le n)$. Let $S_i$ be the Bubble Gum specialty store in city $i$. In each $S_i$, the amount of Bubble Gums expected to be sold in the tourist season is analyzed and reported to the supplier in the form of $[l_i, m_i]$. Here, $l_i$ and $m_i$ represent the minimum and the maximum number of products required at $S_i$, respectively.

The Bubble Gum supply company in city 0 collects the reports from the stores in every city, and supplies products according to the rules described below.

Select a city, say city $k$ $(1 \le k \le n)$. Then, take a train departing from city 0, travel to city $k$, and supply Bubble Gums only to the stores along the route. In other words, the Bubble Gum supplier supplies products to $S_1, S_2, \ldots, S_k$. Let $c_i$ be the number of Bubble Gums supplied to $S_i$ $(1 \le i \le k)$ while moving along the route. Then the condition $c_i \le c_{i+1}$ $(1 \le i \le k-1)$ must be satisfied.

If the supplier supplies products according to the supply rules described above, it may be impossible for every store to get the desired number of products with a single supply procedure. Therefore, the supplier will go through several supply procedures to supply the products, but must comply with the supply rules described above each time. After completing all supply procedures, each $S_i$ must have at least $l_i$ and at most $m_i$ products.

For example, suppose $n = 4$, and the number of products required by each store $S_i$ $(1 \le i \le 4)$ are $[13, 15]$, $[5, 8]$, $[6, 14]$, and $[3, 7]$, respectively. In order for each store to get the desired quantity of goods, there must be at least two supply procedures. In the first supply procedure, 6 products can be supplied to each of the 4 stores. Once this first procedure is completed, all stores' requests except $S_1$ are satisfied. Since 6 products have already been supplied to $S_1$, $r$ $(7 \le r \le 9)$ additional products will be supplied to $S_1$ in the second supply procedure. Of course, there may be other possible schedules. However, at least two supply procedures are required.

Write a program to calculate the minimum number of supply procedures needed to supply the number of Bubble Gums required by each store according to the above rules.

## Input

The input starts with a line containing an integer $n$ $(1 \le n \le 10^6)$ which is the number of cities along the coast.

In the following $n$ lines, the $i$-th line contains two integers $l_i$ and $m_i$ $(1 \le l_i \le m_i \le 10^9)$ which indicate the minimum and the maximum number of products required at $S_i$.

## Output

Print exactly one line. The line should contain the minimum number of supply procedures needed to supply the number of products required by each store according to the supply rules.

## Examples

| *standard input* | *standard output* |
|---|---|
| 4<br>13 15<br>5 8<br>6 14<br>3 7 | 2 |
| 5<br>1 2<br>2 3<br>33 44<br>4 5<br>6 7 | 2 |
| 5<br>10 20<br>3 6<br>13 30<br>7 8<br>11 13 | 3 |