A. Simulating Move to Front

Input file:	`standard input`
Output file:	`standard output`
Time limit:	3 seconds
Memory limit:	256 megabytes

Start with a list $[0, 1, 2, \ldots, n-1]$. Now you get a sequence of m **move to front** requests, where `mtf(`i`)` moves element i to the front of the list. When it does the move, your program should output the index of the element being moved. (Zero-based indexing is used, so the front of the list has index 0.)

In this way, a sequence of m numbers in the range $[0, n-1]$ is transformed into another such sequence. This transformation (and its inverse) is useful in implementing a certain data compression algorithm.

So, for example, say $n = 5$. The initial list is $[0, 1, 2, 3, 4]$. If `mtf(3)` is executed, then the list becomes $[3, 0, 1, 2, 4]$, and 3 is output. If the next request is 3, then the list stays the same and 0 is output. Finally if the next request is `mtf(4)` then the list becomes $[4, 3, 0, 1, 2]$ and 4 is output.

Input

The first line contains blank-separated numbers n and m with $1 \le n \le 10^6$ and $1 \le m \le 5 \times 10^5$. The second line consists of the numbers $p_1, p_2, \ldots, p_m$, each of which is in the range $[0, n-1]$. These are the items to which the move-to-front operation is applied.

Output

The output consists of one line containing m space-separated numbers. The ith of these is the index of the item p_i when it is requested.

Examples

standard input	standard output
5 3 3 3 4	3 0 4
6 8 5 0 4 0 3 0 2 0	5 1 5 1 5 1 5 1

B. Circular RMQ

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

You are given circular array $a_0, a_1, ..., a_{n-1}$. There are two types of operations with it:

- $inc(lf, rg, v)$ — this operation increases each element on the segment $[lf, rg]$ (inclusively) by v;
- $rmq(lf, rg)$ — this operation returns minimal value on the segment $[lf, rg]$ (inclusively).

Assume segments to be circular, so if $n = 5$ and $lf = 3, rg = 1$, it means the index sequence: $3, 4, 0, 1$.

Write program to process given sequence of operations.

Input

The first line contains integer n ($1 \le n \le 200000$). The next line contains initial state of the array: $a_0, a_1, ..., a_{n-1}$ ($-10^6 \le a_i \le 10^6$), a_i are integer. The third line contains integer m ($0 \le m \le 200000$), m — the number of operartons. Next m lines contain one operation each. If line contains two integer lf, rg ($0 \le lf, rg \le n - 1$) it means rmq operation, it contains three integers lf, rg, v ($0 \le lf, rg \le n - 1; -10^6 \le v \le 10^6$) — inc operation.

Output

For each rmq operation write result for it. Please, do not use `%lld` specificator to read or write 64-bit integers in C++. It is preffered to use `cout` (also you may use `%I64d`).

Examples

input
4 1 2 3 4 4 3 0 3 0 -1 0 1 2 1

output
1 0 0

C. Physical Education Lessons

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

This year Alex has finished school, and now he is a first-year student of Berland State University. For him it was a total surprise that even though he studies programming, he still has to attend physical education lessons. The end of the term is very soon, but, unfortunately, Alex still hasn't attended a single lesson!

Since Alex doesn't want to get expelled, he wants to know the number of working days left until the end of the term, so he can attend physical education lessons during these days. But in BSU calculating the number of working days is a complicated matter:

There are n days left before the end of the term (numbered from 1 to n), and initially all of them are working days. Then the university staff sequentially publishes q orders, one after another. Each order is characterised by three numbers l, r and k:

- If $k = 1$, then all days from l to r (inclusive) become non-working days. If some of these days are made working days by some previous order, then these days still become non-working days;
- If $k = 2$, then all days from l to r (inclusive) become working days. If some of these days are made non-working days by some previous order, then these days still become working days.

Help Alex to determine the number of working days left after each order!

Input

The first line contains one integer n, and the second line — one integer q ($1 \le n \le 10^9$, $1 \le q \le 3 \cdot 10^5$) — the number of days left before the end of the term, and the number of orders, respectively.

Then q lines follow, i-th line containing three integers l_i, r_i and k_i representing i-th order ($1 \le l_i \le r_i \le n$, $1 \le k_i \le 2$).

Output

Print q integers. i-th of them must be equal to the number of working days left until the end of the term after the first i orders are published.

Example

input
4 6 1 2 1 3 4 1 2 3 2 1 3 2 2 4 1 1 4 2
output
2 0 2 3 1 4

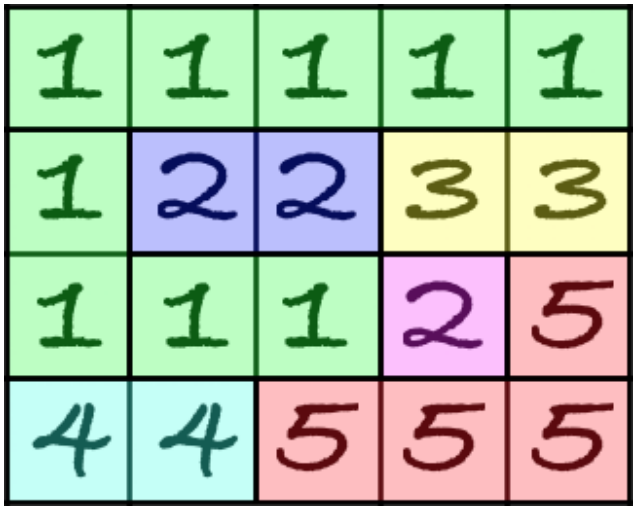
D. Vladik and Entertaining Flags

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

In his spare time Vladik estimates beauty of the flags.

Every flag could be represented as the matrix $n \times m$ which consists of positive integers.

Let's define the beauty of the flag as number of components in its matrix. We call component a set of cells with same numbers and between any pair of cells from that set there exists a path through adjacent cells from same component. Here is the example of the partitioning some flag matrix into components:



But this time he decided to change something in the process. Now he wants to estimate not the entire flag, but some segment. Segment of flag can be described as a submatrix of the flag matrix with opposite corners at $(1, l)$ and (n, r), where conditions $1 \le l \le r \le m$ are satisfied.

Help Vladik to calculate the beauty for some segments of the given flag.

Input

First line contains three space-separated integers n, m, q $(1 \le n \le 10, 1 \le m, q \le 10^5)$ — dimensions of flag matrix and number of segments respectively.

Each of next n lines contains m space-separated integers — description of flag matrix. All elements of flag matrix is positive integers not exceeding 10^6.

Each of next q lines contains two space-separated integers l, r $(1 \le l \le r \le m)$ — borders of segment which beauty Vladik wants to know.

Output

For each segment print the result on the corresponding line.

(examples on next page)

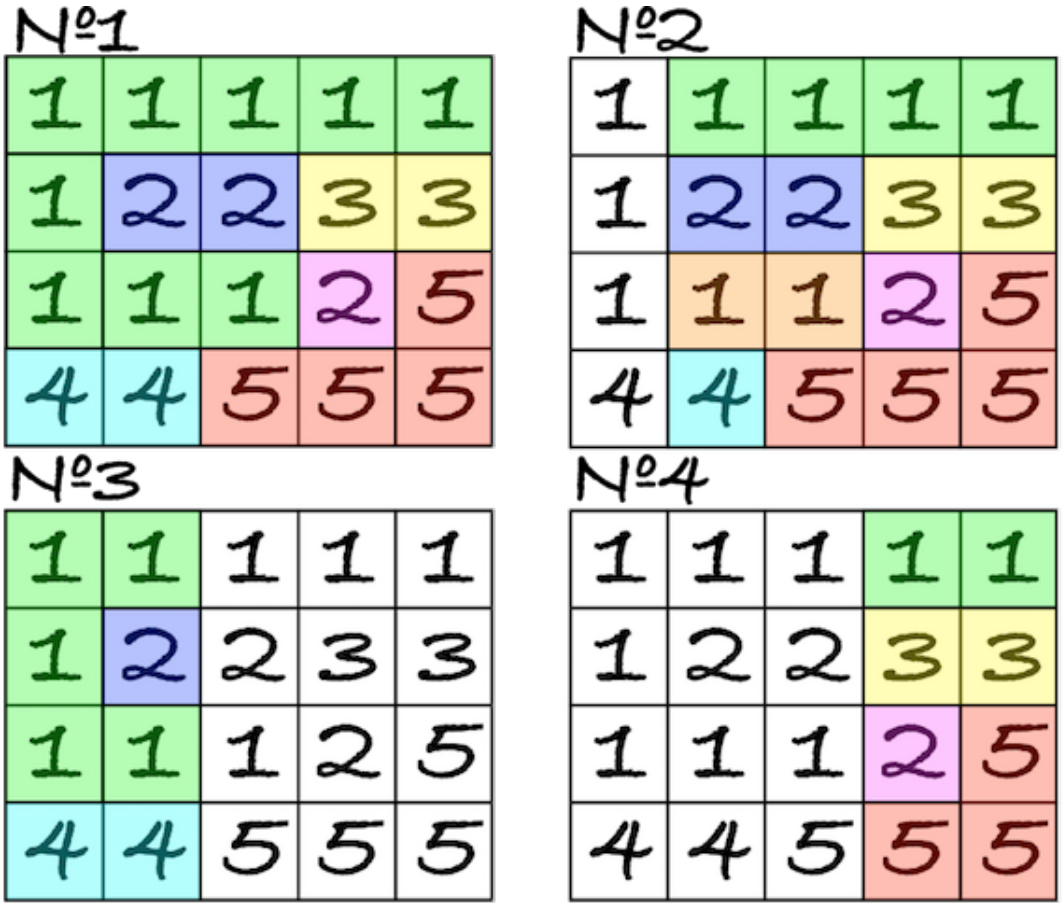
Example

input
4 5 4 1 1 1 1 1 1 2 2 3 3 1 1 1 2 5 4 4 5 5 5 1 5 2 5 1 2 4 5

output
6 7 3 4

Note

Partitioning on components for every segment from first test case:



E. Copying Data

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

We often have to copy large volumes of information. Such operation can take up many computer resources. Therefore, in this problem you are advised to come up with a way to copy some part of a number array into another one, quickly.

More formally, you've got two arrays of integers $a_1, a_2, ..., a_n$ and $b_1, b_2, ..., b_n$ of length n. Also, you've got m queries of two types:

1. Copy the subsegment of array a of length k, starting from position x, into array b, starting from position y, that is, execute $b_{y+q} = a_{x+q}$ for all integer q $(0 \le q < k)$. The given operation is correct — both subsegments do not touch unexistent elements.
2. Determine the value in position x of array b, that is, find value b_x.

For each query of the second type print the result — the value of the corresponding element of array b.

Input

The first line contains two space-separated integers n and m $(1 \le n, m \le 10^5)$ — the number of elements in the arrays and the number of queries, correspondingly. The second line contains an array of integers $a_1, a_2, ..., a_n$ $(|a_i| \le 10^9)$. The third line contains an array of integers $b_1, b_2, ..., b_n$ $(|b_i| \le 10^9)$.

Next m lines contain the descriptions of the queries. The i-th line first contains integer t_i — the type of the i-th query $(1 \le t_i \le 2)$. If $t_i = 1$, then the i-th query means the copying operation. If $t_i = 2$, then the i-th query means taking the value in array b. If $t_i = 1$, then the query type is followed by three integers x_i, y_i, k_i $(1 \le x_i, y_i, k_i \le n)$ — the parameters of the copying query. If $t_i = 2$, then the query type is followed by integer x_i $(1 \le x_i \le n)$ — the position in array b.

All numbers in the lines are separated with single spaces. It is guaranteed that all the queries are correct, that is, the copying borders fit into the borders of arrays a and b.

Output

For each second type query print the result on a single line.

Examples

input
5 10 1 2 0 -1 3 3 1 5 -2 0 2 5 1 3 3 3 2 5 2 4 2 1 1 2 1 4 2 1 2 4 1 4 2 1 2 2

output
0 3 -1 3 2 3 -1

F. New Year and Old Subsequence

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

A string t is called *nice* if a string "2017" occurs in t as a **subsequence** but a string "2016" doesn't occur in t as a **subsequence**. For example, strings "203434107" and "9220617" are nice, while strings "20016", "1234" and "20167" aren't nice.

The *ugliness* of a string is the minimum possible number of characters to remove, in order to obtain a nice string. If it's impossible to make a string nice by removing characters, its ugliness is -1.

Limak has a string s of length n, with characters indexed 1 through n. He asks you q queries. In the i-th query you should compute and print the ugliness of a **substring** (continuous subsequence) of s starting at the index a_i and ending at the index b_i (inclusive).

Input

The first line of the input contains two integers n and q ($4 \leq n \leq 200\,000$, $1 \leq q \leq 200\,000$) — the length of the string s and the number of queries respectively.

The second line contains a string s of length n. Every character is one of digits '0'–'9'.

The i-th of next q lines contains two integers a_i and b_i ($1 \leq a_i \leq b_i \leq n$), describing a substring in the i-th query.

Output

For each query print the ugliness of the given substring.

(see examples on next page)

Examples

input
8 3 20166766 1 8 1 7 2 8
output
4 3 -1

input
15 5 012016662091670 3 4 1 14 4 15 1 13 10 15
output
-1 2 1 -1 -1

input
4 2 1234 2 4 1 2
output
-1 -1

Note

In the first sample:

- In the first query, $ugliness("20166766") = 4$ because all four sixes must be removed.
- In the second query, $ugliness("2016676") = 3$ because all three sixes must be removed.
- In the third query, $ugliness("0166766") = -1$ because it's impossible to remove some digits to get a nice string.

In the second sample:

- In the second query, $ugliness("01201666209167") = 2$. It's optimal to remove the first digit '2' and the last digit '6', what gives a string "010166620917", which is nice.
- In the third query, $ugliness("016662091670") = 1$. It's optimal to remove the last digit '6', what gives a nice string "01666209170".