Problem A. Snowmen

Input:	*standard input*
Output:	*standard output*
Time limit:	4 seconds
Memory limit:	256 mebibytes

It's winter. Year 2222. The news is: cloning of snowmen becomes available.

Snowman consists of zero or more snowballs put one atop another. Each snowball has some mass. Cloning of a snowman produces its exact copy.

Andrew initially has one empty snowman and performs a sequence of the following operations. Clone one of his snowmen and

- either put a new snowball on the top of the new snowman;

- or remove the topmost snowball from the new snowman (the new snoman must be nonemtpy).

He wants to know the total mass of all his snowmen after he performs all operations.

Input

The first line of input contains an integer n $(1 \leq n \leq 200\,000)$. The following lines describe operations, the i-th operation is on of the following:

- `t m` — clone the snowman number t $(0 \leq t < i)$ to get the snowman number i, and put the ball with the mass m on the top of the snowman number i $(0 < m \leq 1000)$;

- `t 0` — clone the snowman number t $(0 \leq t < i)$ to get the snowman number i and remove topmost snowball. It is guaranteed that the snowman number t is not empty.

All masses are integer.

Output

Output the total mass of all snowmen in the end.

Example

standard input	standard output
8 0 1 1 5 2 4 3 2 4 3 5 0 6 6 1 0	74

Problem B. Persistent Queue

Input:	*standard input*
Output:	*standard output*
Time limit:	1.5 seconds
Memory limit:	256 mebibytes

Persistent data structures are designed to allow access and modification of any version of data structure. In this problem you are asked to implement persistent queue.

Queue is the data structure that maintains a list of integer numbers and supports two operations: push and pop. Operation push(x) adds x to the end of the list. Operation pop returns the first element of the list and removes it.

In persistent version of queue each operation takes one additional argument v. Initially the queue is said to have version 0. Consider the i-th operation on queue. If it is push(v, x), the number x is added to the end of the v-th version of queue and the resulting queue is assigned version i (the v-th version is not modified). If it is pop(v), the front number is removed from the v-th version of queue and the resulting queue is assigned version i (similarly, version v remains unchanged).

Given a sequence of operations on persistent queue, print the result of all pop operations.

Input

The first line of the input file contains n — the number of operations ($1 \le n \le 200\,000$). The following n lines describe operations. The i-th of these lines describes the i-th operation. Operation push(v, x) is described as "1 v x", operation pop(v) is described as "-1 v". It is guaranteed that pop is never applied to an empty queue. Elements pushed to the queue fit standard signed 32-bit integer type.

Output

For each pop operation print the element that was extracted.

Examples

standard input	standard output
10	1
1 0 1	2
1 1 2	3
1 2 3	1
1 2 4	2
-1 3	4
-1 5	
-1 6	
-1 4	
-1 8	
-1 9	

Problem C. Persistent Array

Input:	*standard input*
Output:	*standard output*
Time limit:	1 second
Memory limit:	256 mebibytes

You are given the initial revision of an array. You have to perform two operations on it.

- `create` i j x ($a_{new} = a_i$; $a_{new}[j] = x$) — create the new revision from the i-th one, assign the j-th element to x, other elements remain the same as in the i-th revision.

- `get` i j (print $a_i[j]$) — report the value of the j-th element of the i-th revision.

Input

Input contains integer n ($1 \le n \le 10^5$), followed by elements of the initial revision of the array. The initial revision has number 1. The number of queries m ($1 \le m \le 10^5$) follows, then m queries. See sample input for queries formatting. The new revision of the array created when there are k revisions, get number $k+1$. All elements of the array are integers from 0 to 10^9, inclusive. Array is indexed from 1 to n, inclusive.

Output

For each "`get`" query output the corresponding element.

Example

standard input	standard output
6	6
1 2 3 4 5 6	5
11	10
create 1 6 10	5
create 2 5 8	10
create 1 5 30	8
get 1 6	6
get 1 5	30
get 2 6	
get 2 5	
get 3 6	
get 3 5	
get 4 6	
get 4 5	

Discover Singapore 2019 by Moscow Workshops
Day 7, Division B, Persistent Data Structures, Saturday,
September 28, 2019

S≡T   Acronis
Schaffhausen
Institute of
Technology

Problem D. Intercity Express

Input:	*standard input*
Output:	*standard output*
Time limit:	5 seconds
Memory limit:	256 mebibytes

Andrew is developing the a system for train ticket sales. He is going to test it on Intercity Express line that connects two large cities and has $n - 2$ intermediate stations, so there are a total of n stations numbered from 1 to n.

Intercity Express train has s seats numbered from 1 to s. In test mode the system has access to a database that contains already sold tickets in direction from station 1 to station n and needs to answer questions whether it is possible to sell a ticket from station a to station b and if so, what is the minimal number of seat that is vacant on all segments between a and b. Initially the system will have read only access, so even if there is a vacant seat, it should report so, but should not modify the data to report it reserved.

Help Andrew to test his system by writing a program that would answer such questions.

Input

The first line of the input file contains n — the number of stations, s — the number of seats and m — the number of already sold tickets ($2 \le n \le 10^9$, $1 \le s \le 100\,000$, $0 \le m \le 100\,000$). The following m lines describe tickets, each ticket is described by c_i, a_i, and b_i — the seat that the owner of the ticket occupies, the station from which the ticket is sold, and the station to which the ticket is sold ($1 \le c_i \le s$, $1 \le a_i < b_i \le n$).

The following line contains q — the number of queries ($1 \le q \le 100\,000$). A special value p must be maintained when reading queries. Initially $p = 0$. The following $2q$ integers describe queries. Each query is described with two numbers: x_i and y_i ($x_i < y_i$). To get cities a and b between which the seat availability is requested use the following formulae: $a = x_i + p$, $b = y_i + p$. The answer to the query is 0 if there is no seat that is vacant on each segment between a and b, or the minimal number of seat that is vacant.

After answering the query, assign the answer for the query to p.

Output
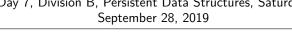
For each query output the answer to it.

Example

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

Note

Note that actual queries are $(1, 2)$, $(2, 3)$, $(3, 4)$, $(4, 5)$, $(1, 3)$, $(2, 4)$, $(3, 5)$, $(1, 4)$, $(2, 5)$, $(1, 5)$.

Problem E. K-th Element on a Segment

Input:	*standard input*
Output:	*standard output*
Time limit:	5 seconds
Memory limit:	512 mebibytes

You are given an array of N non-negative integers, not exceeding 10^9. You have to answer several queries about the k-th element in increasing order on an a segment $[l, r]$.

Input

The first line contains an integer N, the size of the array $(1 \le N \le 450\,000)$.

The second line is used to generate integers a_i, the initial elements of the array $(0 \le a_i < 10^9)$. It contains three integers a_1, l and m $(0 \le a_1, l, m < 10^9)$; for $2 \le i \le N$,

$$a_i = (a_{i-1} \cdot l + m) \bmod 10^9.$$

In particular, $0 \le a_i < 10^9$.

The third line contains an integer B, the number of groups of queries $(1 \le B \le 1000)$. B lines follow, each describing a group of queries. Each set is described by ten integers. The first one is G, the number of queries. Then follow x_1, l_x and m_x, then y_1, l_y and m_y, and finally, k_1, l_k and m_k $(1 \le x_1 \le y_1 \le N, 1 \le k_1 \le y_1 - x_1 + 1, 0 \le l_x, m_x, l_y, m_y, l_k, m_k < 10^9)$. These are used to generate auxiliary sequences x_g and y_g and the actual parameters i_g, j_g and k_g for $1 \le g \le G$:

$$
\begin{aligned}
x_g &= ((i_{g-1} - 1) \cdot l_x + m_x) \bmod N) + 1, & 2 \le g \le G \\
y_g &= ((j_{g-1} - 1) \cdot l_y + m_y) \bmod N) + 1, & 2 \le g \le G \\
i_g &= \min(x_g, y_g), & 1 \le g \le G \\
j_g &= \max(x_g, y_g), & 1 \le g \le G \\
k_g &= (((k_{g-1} - 1) \cdot l_k + m_k) \bmod (j_g - i_g + 1)) + 1, & 1 \le g \le G
\end{aligned}
$$

The generated parameters mean that in g-th query is the k_g-th largest element from the segment $[i_g, j_g]$.

The total number of queries doesn't exceed $600\,000$.

Output

Output one number, the sum of answers to all queries.

Example

standard input	standard output
5	15
1 1 1	
5	
1	
1 0 0 3 0 0 2 0 0	
1	
2 0 0 5 0 0 3 0 0	
1	
1 0 0 5 0 0 5 0 0	
1	
3 0 0 3 0 0 1 0 0	
1	
1 0 0 4 0 0 1 0 0	

Problem F. Rollback

Input:	standard input
Output:	standard output
Time limit:	5 seconds
Memory limit:	256 mebibytes

Sergey has an array of integers $a_1, a_2, \ldots, a_n$, $1 \le a_i \le m$. He wants to answer the following questions: given l what is the minimal r such that there are at least k different values among $a_l, a_{l+1}, \ldots, a_r$.

Input

The first line of input contains two integers: n and m ($1 \le n, m \le 100\,000$). The second line cotnains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le m$).

The following line contains q — the number of queries to answer. ($1 \le q \le 100\,000$). To answer the queries online you must maintain an integer p, initially $p = 0$. Each query is specified with two integers x_i and y_i, use them to get query parameters: $l_i = ((x_i + p) \bmod n) + 1$, $k_i = ((y_i + p) \bmod m) + 1$ ($1 \le l_i, x_i \le n$, $1 \le k_i, y_i \le m$). Let the answer to the i-th query be r_i. After answering the question, set p equal to r_i.

Output

For each query output the minimal value of r_i, of 0 if there is no such r_i.

Example

standard input	standard output
7 3	1
1 2 1 3 1 2 1	4
4	0
7 3	6
7 1	
7 1	
2 2	

Problem G. Urns and Balls

Input:	standard input
Output:	standard output
Time limit:	3 seconds
Memory limit:	256 mebibytes

Consider n different urns and n different balls. Initially, there is one ball in each urn.

There is a special device designed to move the balls. Using this device is simple. First, you choose some range of consecutive urns. The device then lifts all the balls form these urns. After that, you specify the destination which is another range of urns having the same length. The device then moves the lifted balls and places them in the destination urns. Each urn can contain any number of balls.

Given a sequence of movements for this device, find where will each of the balls be placed after all these movements.

Input

The first line contains two integers n and m, the number of urns and the number of movements ($1 \leq n \leq 100\,000$, $1 \leq m \leq 50\,000$).

Each of the next m lines contain three integers $count_i$, $from_i$ and to_i which mean that the device simultaneously moves all balls from urn $from_i$ to urn to_i, all balls from $from_i + 1$ to urn $to_i + 1$, ..., all balls from urn $from_i + count_i - 1$ to urn $to_i + count_i - 1$ ($1 \leq count_i, from_i, to_i \leq n$, $\max(from_i, to_i) + count_i \leq n + 1$).

Output

Output exactly n numbers from 1 to n: the final positions of all balls. The first number is the final position of the ball which was initially in urn 1, the second number is the final position of the ball from urn 2, and so on.

Examples

standard input	standard output
2 3 1 1 2 1 2 1 1 2 1	1 1
10 3 1 9 2 3 7 3 8 3 1	1 2 1 2 3 4 1 2 2 8