Problem A. Binary Search Tree

Time limit:	2 seconds
Memory limit:	256 MiB

Implement binary search tree. We recommend you to implement treap to make sure that it works, do not use `std::set`, you will need treap in further problems.

Input

Input contains descriptions of tree operations, their count doesn't exceed 100000. Each line contains one of the following operations:

- `insert` x — add x to a tree. If there is already x in a tree, don't do anything;
- `delete` x — remove x from a tree. If there is no x in a tree, don't do anything;
- `exists` x — check whether x is in a tree, print "`true`" or "`false`";
- `next` x — print minimal element in a tree greater than x, or "`none`" if there is none;
- `prev` x — print maximal element in a tree less than x, or "`none`" if there is none.

All x are integer not exceeding 10^9.

Output

Print results of `exists`, `next`, `prev`. Follow sample formatting.

Example

standard input	standard output
insert 2 insert 5 insert 3 exists 2 exists 4 next 4 prev 4 delete 5 next 4 prev 4	true false 5 3 none 3

Problem B. Range Minimum Query

Time limit:	2 seconds
Memory limit:	256 MiB

Giggle company opens the new office in Petrozavodsk, and you are invited to the interview. Your task is to solve the following problem.

You have to create a data structure that would maintain a list of integer numbers. All elements of the list are numbered starting from 1. Initially the list is empty. The following two operations must be supported:

- query: "`? i j`" — return the minimal element between the i-th and the j-th element of the list, inclusive;
- modification: "`+ i x`" — add element x after the i-th element of the list. If $i = 0$, the element is added to the front of the list.

Of course, the performance of the data structure must be good enough.

Input

The first line of the input file contains n — the number of operations to perform ($1 \le n \le 200\,000$). The following n lines describe operations. You may consider that no boundary violations occur. The numbers stored in the data structure do not exceed 10^9 by their absolute value.

Output

For each query operation output its result on a line by itself.

Example

standard input	standard output
8 + 0 5 + 1 3 + 1 4 ? 1 2 + 0 2 ? 2 4 + 4 1 ? 3 5	4 3 1

The following table shows the evolution of the list from sample input.

Operation	**The list after the operation**
initially	*empty*
`+ 0 5`	5
`+ 1 3`	5, 3
`+ 1 4`	5, 4, 3
`+ 0 2`	2, 5, 4, 3
`+ 4 1`	2, 5, 4, 3, 1

Problem C. Move to Front

Time limit: 2 seconds
Memory limit: 256 MiB

Corporal Studip likes to give orders to his squad. His favorite order is "move to front". He lines the squad up in a line and gives a series of orders. Each order is: "Soldiers from l_i to r_i — move to front!"

Let us number the soldiers in the initial line up from 1 to n, from left to right. The order "soldiers from l_i to r_i — move to front!" makes the soldiers that are standing at the positions from l_i to r_i inclusive move to the beginning of the line, preserving their order.

For example, if at some moment the soldiers are standing in the following order: $2, 3, 6, 1, 5, 4$, after the order: "soldiers from 2 to 4 — move to front!" the order of soldiers is $3, 6, 1, 2, 5, 4$. If, for example, the order "soldiers from 3 to 4 — move to front!" follows, the new order of soldiers is $1, 2, 3, 6, 5, 4$.

Given the sequence of orders of corporal, find the final line up of the soldiers.

Input

The first line of the input file contains two integer numbers n and m ($2 \le n \le 100\,000$, $1 \le m \le 100\,000$) — the number of soldiers and the number of orders. The following m lines contain orders, each line contains two integer numbers l_i and r_i ($1 \le l_i \le r_i \le n$).

Output

Output n integer numbers — the order of soldiers in the final line up, after executing all orders.

Example

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

Problem D. Key Insertion

Time limit:	2 seconds
Memory limit:	256 MiB

As an employee of the Macrohard Company, you have been asked to implement the new data structure that would be used to store some integer keys.

The keys must be stored in a special ordered collection that can be considered as an array A, which has an infinite number of locations, numbered starting from 1. Initially all locations are empty. The following operation must be supported by the collection: $Insert(L, K)$, where L is the location in the array and K is some positive integer value.

The operation must be processed as follows:

- If $A[L]$ is empty, set $A[L] \leftarrow K$.
- If $A[L]$ is not empty, perform $Insert(L + 1, A[L])$ and after that set $A[L] \leftarrow K$.

Given N integer numbers $L_1, L_2, \ldots, L_N$ you have to output the contents of the array after a sequence of the following operations:
$Insert(L_1, 1)$
$Insert(L_2, 2)$
$\ldots$
$Insert(L_N, N)$

Input

The first line of the input file contains N — the number of $Insert$ operations and M — the maximal position that can be used in the $Insert$ operation ($1 \le N \le 131\,072$, $1 \le M \le 131\,072$).

Next line contains N integer numbers L_i that describe $Insert$ operations to be performed ($1 \le L_i \le M$).

Output

Output the contents of the array after a given sequence of $Insert$ operations. On the first line print W — the number of the greatest location that is not empty. After that output W integer numbers — $A[1], A[2], \ldots, A[W]$. Output zeroes for empty locations.

Example

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

Problem E. Snowmen

Time limit: 1 second
Memory limit: 256 MiB

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 ($1 \le n \le 200\,000$). The following lines describe operations, the i-th operation is on of the following:

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

Examples

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

Problem F. Persistent Queue

Time limit:	1 second
Memory limit:	256 MiB

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 0 1 1 1 2 1 2 3 1 2 4 -1 3 -1 5 -1 6 -1 4 -1 8 -1 9	1 2 3 1 2 4

Problem G. Persistant Array

Time limit:	1 second
Memory limit:	256 MiB

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

Problem H. Intercity Express

Time limit: 3 seconds
Memory limit: 256 MiB

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 1 2 1 2 2 3 -2 0	0
2 4 1 3 1 4 2 5 1 5	0
	0

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 I. Rollback

Time limit:	3 seconds
Memory limit:	256 MiB

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.

Examples

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

Problem J. Urns and Balls

Time limit: 2 seconds
Memory limit: 256 MiB

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

First line contains two integers n and m, the number of urns and the number of movements ($1 \le n \le 100\,000$, $1 \le m \le 50\,000$). 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 $to_i + 1$, $\ldots$, all balls from urn $from_i + count_i - 1$ to urn $to_i + count_i - 1$ ($1 \le count_i, from_i, to_i \le n$, $\max(from_i, to_i) + count_i \le n + 1$).

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