Problem A. Botvinnik and Fisher

Input file:	`chess.in`
Output file:	`chess.out`
Time limit:	15 seconds
Memory limit:	128 Megabytes

The sixth World Champion in Chess Mikhail Botvinnik and the eleventh World Champion in Chess Robert Fisher are playing an unofficial match. It is the hundredth game of the match. Hundreds of moves have been already done... After the tenth adjournment Mikhail and Robert decided to finish the game as fast as possible, because the result is nothing comparing to the friendship between two greatest countries, and chess are so boring today... Luckily, there are only four pieces left on the board. Each side has only one heavy piece (Rook or Queen) and one King, of course. The game is considered finished if one of the sides is checkmated, stalemated or only two Kings left (the first case means that the checkmated side loses, the other two lead to a draw). **No other rules** such as third time repetition, fifty moves rule, etc. are applied to conclude the end of the game.

Your task is to find the fastest end of the game.

Note

Brief explanation of standard chess rules you may need follows. If you are already familiar with chess rules, you may completely skip this note.

- Game is played on a board 8×8, called *chessboard*. The vertical lines are labeled from `a` to `h` (from left to right), the horizontal lines are numbered from `1` to `8` (from down to up).
- All pieces may have one of two colors: white or black. First player owns white pieces, the second owns black ones.
- Players move in turn. The side is not allowed to skip its turn.
- The Queen moves to any non-zero number of cells in horizontal, vertical or diagonal direction.
- The Rook moves to any non-zero number of cells in horizontal or vertical direction.
- The King moves only one cell in horizontal, vertical or diagonal direction.
- A piece is not allowed to jump over any other piece.
- A piece is not allowed to walk into the same cell where another piece of the same color is already located. If a piece moves into the cell with a piece of another color, the moved piece *takes* the piece which stood. The taken piece is removed from the board.
- If one side has a possibility to take the piece of the other one, the piece which could be taken is called *attacked*. If the King is attacked, the situation is called *check*.
- King is never allowed to move into the cell where it will be attacked. It is easy deducible from this fact that two Kings of different colors cannot be neighbors. Also it is evident that the King may only be checked if it is the move of the side which owns the King.
- If the side is checked, it must make its King to be not checked. It could be made with moving the King off the check, closing the line of checking or taking the piece which checked the King.
- If the King is checked and the corresponding side cannot move, the situation is called *mate* or *checkmate*.
- If the side is not checked and has no available moves, and it is the turn of this side, the situation is called *stalemate*.
- If there are only two Kings left on the board, the game is considered to be ended in a draw.

Input

The first line contains white pieces description, the second one is for black pieces. Each piece is determined by three characters. The first is the type of the piece: `K` for King, `Q` for Queen and `R` for Rook. The second and third characters are coordinates of the field, a letter corresponding to a vertical line and then the digit corresponding to a horizontal one. For example, `Ka1` means that the King stands in left-down corner of the chessboard. The pieces within the single line are separated by space. The position is guaranteed to be valid and it is always turn of the black side. The King is always given first for each side.

Output

Output the shortest valid end of the game. After doing these moves, the position must correspond to one of three cases listed above. Each move is displayed using six characters: first corresponds to piece that moves, second and third correspond to source position, the fourth character must be always a dash ('-'), the fifth and sixth show the destination position of the piece. If there are several optimal solutions, any will do.

Example

chess.in	chess.out
Ka1 Qb5 Ka3 Qa8	Qa8-a4 Qb5-b2
Ka1 Qf5 Ka3 Rh8	Rh8-f8 Qf5-f1 Rf8-f1

Problem B. Deletion

Input file:	`deletion.in`
Output file:	`deletion.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

Vasya deletes a string character by character, one character per second. But the order of characters deleted is not so trivial. At first, Vasya deletes k-th, $2k$-th, ... ik-th characters of the original string, until ik is less than length of the original string. Then Vasya considers the resulting string after all those deletions and does the same with it, and so on, until the resulting string contains less than K characters.

For example, if Vasya had the string of length 10 and $K = 2$, in the first five seconds the characters at positions 2, 4, 6, 8 and 10 will be deleted. In the next two seconds Vasya deletes 2nd and 4th characters of the new string which in fact are 3th and 7th characters of the original one. Then the 2nd character of the third-generation string will be deleted, which is the 5th character of the original one. The last character to be deleted is the 9th character of the original string.

Your task is to write a program which will determine the number of second at which some character of the original string is deleted.

Input

The first line contains three integers: $1 \leqslant n \leqslant 5 \cdot 10^6$ — n is the length of original string, $1 \leqslant k \leqslant n$ is decribed above and $1 \leqslant l \leqslant 10\,000$ — number of queries to solve. l lines follow, containing the queries — the positions of the characters of the original string, in range of $1..n$. Queries may repeat.

Output

For each query output on the separate line the number of second the corresponding character will be deleted, if it will not be deleted at all, output zero for the query.

Example

deletion.in	deletion.out
10 2 6 1 2 3 5 10 5	0 1 6 8 5 8

Problem C. Intelligence

Input file:	intel.in
Output file:	intel.out
Time limit:	2 seconds
Memory limit:	64 Megabytes

The Dogland Intelligence Service has a wide network of agents in Catland. Of course, the Catland Counter-Intelligence wants to have a way to send false information into this network. Now after many preparations they can persuade any of Dogland agents to work for the Catland.

But any Dogland agent working for the Catland will require much expenses, so the Intelligence Service wishes to find a plan in which the number of Dogland agents to recruit will be as small as possible.

The Dogland network consists of two kinds of agents: spies and radio operators. Spies collect information in the Catland, and radio operators transmit it to the Dogland. The lists of radio opearators which may communicate with each spy is known to the Catland Intelligence Service.

Catland Intelligence Service has also information about the present state of Dogland agent network. Each spy may have at most one *current radio operator* in each time. Other radio operators are declared reserve radio operators for this spy and communications with them are used under unexpected circumstances. No two spies may have the same current radio operator. Catland Intelligence Service discovered that Dogland Network is now built optimally according to this rule, i. e. the number of spies which have the current radio operator is maximal possible.

Your task is to find the minimal number of Dogland agents to recruit and a possible plan of recruiting such that any information transmitted via each communication (current or reserve) will reach the Dogland already falsified on some side.

Input

The first line of the input file contains m, n ($1 \leqslant m \leqslant 100\,000$, $1 \leqslant n \leqslant 100\,000$) and c ($0 \leqslant c \leqslant 500\,000$) where m is the number of spies, n is the number of radio operators and c is the number of links between them. c lines follow, each cotaining two numbers u_i and v_i ($0 \leqslant u_i \leqslant m-1$, $0 \leqslant v_i \leqslant n-1$) describing the spy u_i could communicate with radio operator v_i. The last line of the input file contains the list of current operators. It consists of m numbers $-1 \leqslant L_i \leqslant n-1$ — the number of current radio operator for each spy, or -1 if the spy does not have a current radio operator.

Output

The first line must contain a minimal number of Dogland agents to recruit. The second line contains number of spies to recruit S, followed by S identifires of spies in ascending order. The third line contains the list of radio operators to recruit in similar format.

Example

intel.in	intel.out
3 2 4 1 1 0 1 2 1 0 0 0 1 -1	2 1 0 1 1

Problem D. Key Insertion

Input file:	key.in
Output file:	key.out
Time limit:	2 seconds
Memory limit:	256 mebibytes

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:

$Insert(L_1, 1)$
$Insert(L_2, 2)$
...
$Insert(L_N, N)$

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 ($1 \leqslant N \leqslant 131\,072$, $1 \leqslant M \leqslant 131\,072$).

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

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

key.in	key.out
5 4 3 3 4 1 3	6 4 0 5 2 3 1

Problem E. Move to Front

Input file:	movetofront.in
Output file:	movetofront.out
Time limit:	4 seconds
Memory limit:	256 megabytes

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 ($2 \leqslant n \leqslant 100\,000$, $1 \leqslant m \leqslant 100\,000$) — 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 ($1 \leqslant l_i \leqslant r_i \leqslant n$).

Output

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

Example

movetofront.in	movetofront.out
6 3 2 4 3 5 2 2	1 4 5 2 3 6

Problem F. Permuatations Strike Back

Input file:	permutation2.in
Output file:	permutation2.out
Time limit:	5 seconds
Memory limit:	256 mebibytes

Initially you are given an array of integer numbers P_i which in fact is the permutation of the numbers $1..N$. Your task is to write a program which will answer the queries: for $x \leqslant j \leqslant y$, how many P_j are between k and l, inclusive, and will support the arbitrary changes of the elements of the array (so it is not always a permutation).

Input

First line contains two integers: $1 \leqslant N \leqslant 100\,000$ — the size of the initial permutation and $1 \leqslant M \leqslant 100\,000$ — the number of queries. The second line contains N numbers — the permutation itself. M lines follow, one for each query. A query can be in form `SET a b` ($1 \leqslant a \leqslant N$, $1 \leqslant b \leqslant N$). Set query changes a number in position a to be equal to b. Also the query can be in form `GET x y k l` ($1 \leqslant x \leqslant y \leqslant N$, $1 \leqslant k \leqslant l \leqslant N$) — just GET query as described above.

Output

For each GET query output the answer on a separate line.

Example

permutation2.in	permutation2.out
4 4 1 2 3 4 GET 1 2 2 3 GET 1 3 1 3 SET 1 4 GET 1 3 1 3	1 3 2

Problem G. Reversing the Segments

Input file:	reverse.in
Output file:	reverse.out
Time limit:	6 seconds
Memory limit:	256 mebubytes

There is an array of n numbers. You need to support two kinds of events:

1. query of sum of all numbers standing from l-th to r-th, inclusive
2. reversing the segment from l-th to r-th, inclusive, that means l-th number swaps with r-th, $(l+1)$-th with $(r-1)$-th and so on.

Input

The first line contains two integers n and m ($1 \leqslant n, m \leqslant 200\,000$) — the counts of numbers and events respectively. The second lines contains initial state of array — n positive integers that are not greater than $2 \cdot 10^5$. m lines follow, each describing one event by three integers q, l, r ($0 \leqslant q \leqslant 1$, $1 \leqslant l \leqslant r \leqslant n$). q is 0 for sum query, 1 for reverse query.

Output

For each sum query just output the answer.

Example

reverse.in	reverse.out
5 6 1 2 3 4 5 0 1 5 0 2 4 1 2 4 0 1 3 0 4 5 0 3 5	15 9 8 7 10

Problem H. Just Sum the Numbers

Input file:	sum.in
Output file:	sum.out
Time limit:	5 seconds
Memory limit:	256 mebibytes

You need to implement the data structure which supports the following operations on a set S of integer numbers:

- $add(i)$ — add i to S (if i already belongs to S, S is not changed)
- $sum(l, r)$ — sum all x from S, which satisfy $l \leqslant x \leqslant r$.

Input

Initially S is empty. The first line contains count of operations n ($1 \leqslant n \leqslant 300\,000$). Each of the following n lines describes one operation. Operation is described either by «`+ i`» or «`? l r`». Operation «`? l r`» stands for query $sum(l, r)$.

Operation «`+ i`» is interpreted depending of its previous operation. If it's the first operation in the input or it follows another «`+`» operation, it's just $add(i)$. But if it follows some query «`?`» with the result y, it must be interpreted as $add((i + y) \bmod 10^9)$.

Parameters of all operations are between 0 and 10^9, inclusive.

Output

Output the answers to all queries.

Example

sum.in	sum.out
6 + 1 + 3 + 3 ? 2 4 + 1 ? 2 4	3 7

Problem I. Swapper

Input file:	swapper.in
Output file:	swapper.out
Time limit:	5 seconds
Memory limit:	256 mebibytes

You need to build an effective model of the data structure which can do the following with an array of integer numbers:

- Take a segment of an even length from x-th to y-th number, inclusive, and swap x-th with $(x+1)$-th, $(x+2)$-th with $(x+3)$-th and so on
- Find the sum of the numbers of any segment from x-th to y-th numbers, inclusive.

Input

One or more cases, first line of each case contains two integers N — the length of array and M — the number of events ($1 \leqslant N, M \leqslant 10^5$). The total sum of all N's over the input does not exceed 200 000. The total sum of all M's over the input also does not exceed 200 000. The second line of each case contains the initial state of an array — N numbers with an absolute value up to 10^9. The last M lines of the case are queries in either in form 1 x y (swap queries) or 2 x y (sum queries). Input is terminated by two zeroes.

Output

Adhere to the sample output below. For each case output the answers to the sum queries. Separate answers to different cases with an empty line.

Examples

swapper.in	swapper.out
5 5 1 2 3 4 5 1 2 5 2 2 4 1 1 4 2 1 3 2 4 4 0 0	Swapper 1: 10 9 2