## Problem A. Sum

| | |
|---|---|
| Input file: | `sum.in` |
| Output file: | `sum.out` |
| Time limit: | 2 seconds |
| Memory limit: | 256 megabytes |

Consider an array of $n$ elements. Your task is to find sum of the elements in a given range.

### Input

The first line of the input contains two integers $n$ and $k$ ($1 \le n \le 100\,000$, $1 \le k \le 100\,000$) — the size of the array and the number of queries, respectively. Each of the following $k$ lines contains a description of the query:

- $A\,i\,x$ — assign to the $i$-th element of the array value $x$ ($1 \le i \le n$, $0 \le x \le 10^9$);
- $Q\,l\,r$ — find the sum of elements of the array in range from $l$ to $r$ ($1 \le l \le r \le n$).

Initially the array is initialized with zeros.

### Output

For every query in format $Q\,l\,r$ output a single number — sum of the elements in the specified range.

### Example

| sum.in | sum.out |
|---|---|
| 5 9<br>A 2 2<br>A 3 1<br>A 4 2<br>Q 1 1<br>Q 2 2<br>Q 3 3<br>Q 4 4<br>Q 5 5<br>Q 1 5 | 0<br>2<br>1<br>2<br>0<br>5 |

## Problem B. Range Variation Query

| | |
|---|---|
| Input file: | `rvq.in` |
| Output file: | `rvq.out` |
| Time limit: | 2 seconds |
| Memory limit: | 256 megabytes |

Consider the sequence $a_n$. Initially it is defined with the following formula: $a_n = n^2 \mod 12345 + n^3 \mod 23456$. Your task is to respond to the following queries:

- find difference between the maximal and minimal values among the elements $a_i, a_{i+1}, \ldots, a_j$;
- assign to the element $a_i$ the value $j$.

### Input

The first line of the input contains two integers $n$ and $k$ ($1 \le n \le 100\,000$, $1 \le k \le 100\,000$) — the number of elements in the array and the number of queries, respectively. Each of the following $k$ lines contains a description of a query: two integers $x_i, y_i$.

If $x_i > 0$ then your task is to find the difference between the maximal and minimal values among the elements $a_{x_i}, \ldots, a_{y_i}$. In this case $1 \le x_i \le y_i \le 100\,000$.

If $x_i < 0$ then assign to the element $a_{|x_i|}$ the value $y_i$. In this case $-100\,000 \le x \le -1$ and $y \le 100\,000$.

### Output

For every query of first kind output a single number — difference between maximal and minimal value in the corresponding range.

### Example

| rvq.in | rvq.out |
|---|---|
| 7<br>1 3<br>2 4<br>-2 -100<br>1 5<br>8 9<br>-3 -101<br>2 3 | 34<br>68<br>250<br>234<br>1 |
| 5 3<br>5 4 3 2 1 | 10 |

## Problem C. Sum 2

| | |
|---|---|
| Input file: | `sum.in` |
| Output file: | `sum.out` |
| Time limit: | 2 seconds |
| Memory limit: | 256 megabytes |

Consider an array of $n$ elements. Your task is to find sum of the elements in a given range.

### Input

The first line of the input contains two integers $n$ and $k$ ($1 \le n \le 100\,000$, $1 \le k \le 100\,000$) — the number of elements in the array and the number of queries, respectively. Each of the following $k$ lines contains a description of the query:

- $A\,l\,r\,x$ — assign to the elements in the range from $l$ to $r$ value $x$ ($1 \le l \le r \le n$, $0 \le x \le 10^9$);
- $Q\,l\,r$ — find the sum of the elements of the array from $l$ to $r$ ($1 \le l \le r \le n$).

Initially the array is initialized with zeros.

### Output

For every query in format $Q\,l\,r$ output a single number — the sum of elements in the specified range.

### Example

| sum.in | sum.out |
|---|---|
| 5 9<br>A 2 3 2<br>A 3 5 1<br>A 4 5 2<br>Q 1 3<br>Q 2 2<br>Q 3 4<br>Q 4 5<br>Q 5 5<br>Q 1 5 | 3<br>2<br>3<br>4<br>2<br>7 |

## Problem D. RMQ

| | |
|---|---|
| Input file: | `rmq.in` |
| Output file: | `rmq.out` |
| Time limit: | 1 second |
| Memory limit: | 256 megabytes |

Consider an array of $n$ elements. Your task is to write a program, that processes two kinds of queries:

- $max\,l\,r$ — find the maximum in the array from $l$ to $r$ inclusive ($1 \le l \le r \le 10^5$);
- $add\,l\,r\,v$ — add the value $v$ to all elements of the array from $l$ to $r$ inclusive ($1 \le l \le r \le 10^5$, $|v| \le 10^5$).

### Input

The first line of the input contains two integers $n$ and $q$ ($1 \le n, q \le 10^5$) — the number of the elements in the array and the number of queries, respectively. The second line contains $n$ integer numbers $a_1, \ldots, a_n$ ($|a_i| \le 10^5$) — the elements of the array. Each of the following $q$ lines contains a description of the query in format described above.

### Output

For every query in format $Q\,l\,r$ output a single number — the sum of the elements in the range.

### Example

| rmq.in | rmq.out |
|---|---|
| 5 3<br>1 2 3 4 -5<br>max 1 3<br>add 1 2 5<br>max 1 3 | 3<br>7 |

## Problem E. Signchange

| | |
|---|---|
| Input file: | signchange.in |
| Output file: | signchange.out |
| Time limit: | 2 seconds |
| Memory limit: | 256 megabytes |

Consider an array of $n$ elements. Your task is to implement the data structure, that allow the following queries:

- set element $a_i$ to the value $j$;
- find the alternating sum of the elements of the array from $l$ to $r$ inclusive ($a_l - a_{l+1} + a_{l+2} - \cdots \pm a_r$).

### Input

The first line contains single integer $n$ ($1 \le n \le 10^5$) — the length of the array. The second line contains $n$ integer numbers $a_1, \ldots, a_n$ ($|a_i| \le 10^5$) — the elements of the array. The third line contains an integer $m$ ($1 \le n \le 10^5$) — the number of queries:

- the query of the first kind is defined with three integers: $0\,i\,j$ ($1 \le i \le n$, $1 \le j \le 10^4$);
- the query of the second kind is defined with three integers: $1\,l\,r$ ($1 \le l \le r \le n$).

### Output

For every query of second kind output a single number — the corresponding alternating sum.

### Example

| signchange.in | signchange.out |
|---|---|
| 3<br>1 2 3<br>5<br>1 1 2<br>1 1 3<br>1 2 3<br>0 2 1<br>1 1 3 | -1<br>2<br>-1<br>3 |

## Problem F. $K$-inversions

| | |
|---|---|
| Input file: | kinverse.in |
| Output file: | kinverse.out |
| Time limit: | 2 seconds |
| Memory limit: | 256 megabytes |

Consider a permutation $a_1, a_2, \ldots, a_n$ (all $a_i$ are different integers in range from 1 to $n$). Let us call *$k$-inversion* a sequence of numbers $i_1, i_2, \ldots, i_k$ such that $1 \le i_1 < i_2 < \ldots < i_k \le n$ and $a_{i_1} > a_{i_2} > \ldots > a_{i_k}$. Your task is to evaluate the number of different $k$-inversions in a given permutation.

### Input

The first line of the input contains two integers $n$ and $k$ ($1 \le n \le 20\,000$, $2 \le k \le 10$). The second line is filled with $n$ numbers $a_i$.

### Output

Output a single number — the number of $k$-inversions in a given permutation. The number must be taken modulo $10^9$.

### Example

| kinverse.in | kinverse.out |
|---|---|
| 3 2<br>3 1 2 | 2 |
| 5 3<br>5 4 3 2 1 | 10 |

## Problem G. RMQ Inverse Problem

| | |
|---|---|
| Input file: | rmq.in |
| Output file: | rmq.out |
| Time limit: | 2 seconds |
| Memory limit: | 256 megabytes |

Consider an array of $n$ elements. Let $Q(i, j)$ be the response to a query for finding the minimum among elements of the array from $i$ to $j$ inclusive. Your task is to restore an array given some queries and their responses.

### Input

The first line of the input contains two integers $n$ and $m$ ($1 \le n, m \le 100\,000$) — the number of the elements in the array and the number of queries, respectively. Each of the following $m$ lines contains a description of a query: three integers $i, j$ and $q$ denoting that $Q(i, j) = q$ ($1 \le i \le j \le n$, $2^{-31} \le 1 \le 2^{31} - 1$).

### Output

If required array does not exist, output single line which contains the single word “`inconsistent`”. In the other case the first line must contain the single word “`consistent`”. The second line must contain $n$ integers — elements of the required array. All integers must be in the interval from $2^{-31}$ to $2^{31} - 1$ inclusive. If there are different solutions, output any of them.

### Example

| rmq.in | rmq.out |
|---|---|
| 3 2<br>1 2 1<br>2 3 2 | consistent<br>1 2 3 |
| 3 3<br>1 2 1<br>1 1 2<br>2 3 2 | inconsistent |

## Problem H. Bus

| | |
|---|---|
| Input file: | `taxibus.in` |
| Output file: | `taxibus.out` |
| Time limit: | 2 seconds |
| Memory limit: | 256 megabytes |

Usually the residents of Flatland use buses to travel in the city. Unfortunately, there is a problem — buses are designed in the way that passengers feel discomfort passing each other on the way to free seats.

Spiridon works as a bus driver for a long time. And he knows, that all passengers ride on a bus day by day. Therefore he decided to arrange places among passangers, in the way to minimize the number of times, when one passenger passes another to get to his place.

It is known when each passenger enters and leaves bus. All places are arranged in a row and are numbered from 1 to $n$ starting from the closest seat to entry. When passenger sitting on the $i$-th place goes to the exit, he passes all passengers who are sitting on the seats with numbers less than $i$.

During the ride, passengers do not change their place during the ride.

### Input

The first line contains single integer $n$ ($1 \le n \le 100\,000$) — the number of passengers. Each of the following $n$ lines contains two integers $a_i, b_i$ — the indices of bus stops on which the $i$-th passenger enters and leaves bus, respectively. At each bus stop at most one person enters or leaves the bus.

### Output

The first line must contain the single integer — the mimimal number of passes of passengers past each other. The second line must contain $n$ integers — for each passenger output the place on which passenger should sit.

### Example

| taxibus.in | taxibus.out |
|---|---|
| 2<br>1 4<br>2 3 | 0<br>2 1 |
| 5<br>1 8<br>3 6<br>2 4<br>9 10<br>5 7 | 2<br>10 2 4 1 1 |

## Problem I. Rectangles

| | |
|---|---|
| Input file: | `rects.in` |
| Output file: | `rects.out` |
| Time limit: | 2 seconds |
| Memory limit: | 256 megabytes |

Consider $n$ rectangles on the plane such that any two of them do not have common points. A rectangle $B$ is *farther* than a rectangle $A$, if $B$'s top left corner lies strictly below and strictly right than bottom right corner of $A$.

A sequence of rectangles $R_1, R_2, \ldots, R_k$ is called a *chain*, if for all $i$ the rectangle $R_i$ is farther than the rectangle $R_{i-1}$. *Weight* of chain is the sum of the numbers writen in rectangles of this chain.

Your task is to find the chain of rectangles with maximal possible weight.

### Input

The first line contains single integer $n$ ($1 \le n \le 100\,000$) — the number of the rectangles. Let $x$-axis goes from left to right and $y$-axis goes from down to up. Each of the following $n$ lines contains five integers — coordinates $x_{i,1}, y_{i,1}$ of left bottom corner, coordinates $x_{i,2}, y_{i,2}$ top right corner and the number $a_i$ writen in the $i$-th rectangle. All coordinates do not exceed $10^9$ in absolute value. Numbers writen inside rectangles are positive and do not exceed $10^9$. Every rectangle do not lie inside another rectangle.

### Output

The first line must contain a single integer — the maximal possible weight of chain of rectangles. The second line must contain numbers of rectangles forming such chain in the corresponding order. If there are several optimal solutions, output any of them.

### Example

| rects.in | rects.out |
|---|---|
| 4<br>1 1 2 2 6<br>3 1 4 2 5<br>0 3 1 4 5<br>5 1 6 2 4 | 10<br>3 2 |

## Problem J. Windows

| | |
|---|---|
| Input file: | `windows.in` |
| Output file: | `windows.out` |
| Time limit: | 1 second |
| Memory limit: | 256 megabytes |

Consider rectangular windows on the screen with sides parallel to the coordinate axis. Some of them can overlap. Your task is to find a point on screen covered by the maximal number of the windows.

### Input

The first line contains single integer $n$ ($1 \le n \le 50\,000$) — the number of windows. Each of the following $n$ lines contains coordinates of windows $x_{(1,i)}, y_{(1,i)}, x_{(2,i)}, y_{(2,i)}$, where $(x_{(1,i)}, y_{(1,i)})$ are coordinates of the left top corner, and $(x_{(2,i)}, y_{(2,i)})$ are coordinates of the right bottom corner of the $i$-th window (on the computer screen $x$ coordinate grows down and $y$ coordinate grows from left to right). All coordinates are integers and their absolute values do not exceed $10^9$.

### Output

The first line must contain the single integer — the maximal possible number of windows that covered some point in the given configuration. The second line must contain two integer numbers — coordinates of the point covered maximal number of windows. The windows are closed, i.e., window contains its own border points.

### Example

| windows.in | windows.out |
|---|---|
| 2<br>0 0 3 3<br>1 1 4 4 | 2<br>3 2 |

## Problem K. Windows 2

| | |
|---|---|
| Input file: | `windows2.in` |
| Output file: | `windows2.out` |
| Time limit: | 2 seconds |
| Memory limit: | 256 megabytes |

Consider rectangular windows on the screen with sides parallel to the coordinate axis. Some of them can overlap. Your task is to find area covered by this windows.

### Input

The first line contains single integer $n$ ($1 \le n \le 50\,000$) — the number of windows. Each of the following $n$ lines contains coordinates of windows $x_{(1,i)}, y_{(1,i)}, x_{(2,i)}, y_{(2,i)}$, where $(x_{(1,i)}, y_{(1,i)})$ are coordinates of the left top corner, and $(x_{(2,i)}, y_{(2,i)})$ are coordinates of the right bottom corner of the $i$-th window (on the computer screen $x$ coordinate grows down and $y$ coordinate grows from left to right). All coordinates are integers and their absolute values do not exceed $10^9$.

### Output

The single line must contain the single integer — the area covered by windows.

### Example

| windows2.in | windows2.out |
|---|---|
| 2<br>0 0 3 3<br>1 1 4 4 | 14 |

## Problem L. Two Captains

| | |
|---|---|
| Input file: | `twocaptains.in` |
| Output file: | `twocaptains.out` |
| Time limit: | 3 seconds |
| Memory limit: | 256 megabytes |

As it is known, Jack Sparrow and Barbossa are captains of The Black Pearl. The ship carries exactly $n$ cannons, located in a row. During the battle, both captains give orders to their sailors simultaneously each minite. There are three different kinds of orders:

- `send` $l\,r$ — send his sailors to shoot from the cannons with numbers from $l$ to $r$ inclusive;
- `back` $l\,r$ — recall all his sailors from the cannons with numbers from $l$ to $r$ inclusive. If there are no sailors obeying this captain on some cannons from this segment, nothing happens;
- `rum` — bring one more bottle of rum.

Every order is executed immediately and battle continues one more minute until the next order. If at any moment of time sailors obeing different captains stay on the same cannon, they will fight and kill each other. This situation isn't suitable for both captains, so they ask you to help them with solution of this problem.

Before the start of another battle, captain Jack Sparrow and Barbossa make plans of their actions. The plan of captain Jack Sparrow consists of $m_1$ orders and the plan of Barbossa consists of $m_2$ orders. At the beginning of the $i$-th minute of battle, each captain gives his sailors the $i$-th order from his plan if the plan consists of at least $i$ orders. Your task is to fix plans in the way all sailors will stay alive. The only available modification for you is to insert some orders `rum` in any places of plans. Since captains do not enjoy change plans, the total number of additional orders should be minimal.

### Input

The first line contains single integer $n$ ($1 \le n \le 10^9$) — the number of cannons on the ship.

The second line contains an integer $m_1$ ($1 \le m_1 \le 3\,000$) — the number of orders in the plan of Captain Jack Sparrow. Each of the following $m_1$ lines contains description of this orders. The orders are given as described above. For all instructions that used $l$, $r$ $1 \le l \le r \le n$. It is guaranteed that the last order is `back` $1\,n$.

The next line contains an integer $m_2$ ($1 \le m_2 \le 3\,000$) — the number of orders in the plan of Barbossa. Each of the following $m_2$ lines contains description of this orders. The orders are given as described above. For all orders that used $l$, $r$ $1 \le l \le r \le n$. It is guaranteed that the last order is `back` $1\,n$.

### Output

The single line must contain the single integer — the minimal number of additional orders.

### Example

| twocaptains.in | twocaptains.out |
|---|---|
| 3<br>4<br>send 1 1<br>send 2 2<br>back 1 1<br>back 1 3<br>5<br>send 2 3<br>send 1 1<br>back 2 2<br>rum<br>back 1 3 | 3 |