

## Задача А. Снеговики

Имя входного файла: `snowmen.in`  
Имя выходного файла: `snowmen.out`  
Ограничение по времени: 4 секунды  
Ограничение по памяти: 64 мегабайта

Зима. 2012 год. На фоне грядущего Апокалипсиса и конца света незамеченной прошла новость об очередном прорыве в областях клонирования и снеговиков: клонирования снеговиков. Вы конечно знаете, но мы вам напомним, что снеговик состоит из нуля или более вертикально поставленных друг на друга шаров, а клонирование — это процесс создания идентичной копии (клона).

В местечке Местячково учитель Андрей Сергеевич Учитель купил через интернет-магазин «Интернет-магазин аппаратов клонирования» аппарат для клонирования снеговиков. Теперь дети могут играть и даже играют во дворе в следующую игру. Время от времени один из них выбирает понравившегося снеговика, клонирует его и:

- либо добавляет ему сверху один шар;
- либо удаляет из него верхний шар (если снеговик не пустой).

Учитель Андрей Сергеевич записал последовательность действий и теперь хочет узнать суммарную массу всех построенных снеговиков.

### Формат входных данных

Первая строка содержит количество действий  $n$  ( $1 \leq n \leq 200\,000$ ). В строке номер  $i + 1$  содержится описание действия  $i$ :

- $t\ m$  — клонировать снеговика номер  $t$  ( $0 \leq t < i$ ) и добавить сверху шар массой  $m$  ( $0 < m \leq 1000$ );
- $t\ 0$  — клонировать снеговика номер  $t$  ( $0 \leq t < i$ ) и удалить верхний шар. Гарантируется, что снеговик  $t$  не пустой.

В результате действия  $i$ , описанного в строке  $i + 1$  создается снеговик номер  $i$ . Изначально имеется пустой снеговик с номером ноль.

Все числа во входном файле целые.

### Формат выходных данных

Выведите суммарную массу построенных снеговиков.

### Примеры

<code>snowmen.in</code>	<code>snowmen.out</code>
8	74
0 1	
1 5	
2 4	
3 2	
4 3	
5 0	
6 6	
1 0	

## Задача В. НВП на дереве

Имя входного файла: стандартный ввод  
Имя выходного файла: стандартный вывод  
Ограничение по времени: 2 секунды  
Ограничение по памяти: 256 мегабайт

Дано дерево на  $N$  вершинах, у которого  $i$ -е ребро соединяет вершину  $u_i$  и вершину  $v_i$ . На вершине с номером  $i$  записано целое число  $a_i$ . Для каждого целого числа  $k$  от 1 до  $N$  решите следующую задачу:

Составим последовательность, выписав целые числа на вершинах, вдоль кратчайшего пути от вершины 1 до вершины  $k$ , в том порядке, в котором они появляются. Найдите длину наибольшей возрастающей подпоследовательности этой последовательности.

### Формат входных данных

В первой строке вводится число  $n$  ( $2 \leq n \leq 2 \cdot 10^5$ ) — количество вершин в дереве.

Во второй строке через пробел задаются числа  $a_i$  ( $1 \leq a_i \leq 10^9$ ) — числа, записанные на вершинах.

В каждой из следующих  $n - 1$ -й строке вводятся пары  $v_i, u_i$  — рёбра дерева ( $1 \leq v_i \neq u_i \leq n$ ). Гарантируется, что данный набор рёбер образует дерево.

### Формат выходных данных

Выведите  $N$  строк. В  $k$ -й строке выведите длину наибольшей возрастающей подпоследовательности последовательности, полученной по кратчайшему пути из вершины 1 в вершину  $k$ .

### Примеры

стандартный ввод	стандартный вывод
10	1
1 2 5 3 4 6 7 3 2 4	2
1 2	3
2 3	3
3 4	4
4 5	4
3 6	5
6 7	2
1 8	2
8 9	3
9 10	

## Задача C. Persistent Array

Имя входного файла: стандартный ввод  
Имя выходного файла: стандартный вывод  
Ограничение по времени: 2 секунды  
Ограничение по памяти: 256 мегабайт

Дан массив (вернее, первая, начальная его версия).

Нужно уметь отвечать на два запроса:

- о  $a_i[j] = x$  — создать из  $i$ -й версии новую, в которой  $j$ -й элемент равен  $x$ , а остальные элементы такие же, как в  $i$ -й версии.
- о `get  $a_i[j]$`  — сказать, чему равен  $j$ -й элемент в  $i$ -й версии.

### Формат входных данных

Количество чисел в массиве  $N$  ( $1 \leq N \leq 10^5$ ) и  $N$  элементов массива. Далее количество запросов  $M$  ( $1 \leq M \leq 10^5$ ) и  $M$  запросов. Формат описания запросов можно посмотреть в примере. Если уже существует  $K$  версий, новая версия получает номер  $K + 1$ . И исходные, и новые элементы массива — целые числа от 0 до  $10^9$ . Элементы в массиве нумеруются числами от 1 до  $N$ .

### Формат выходных данных

На каждый запрос типа `get` вывести соответствующий элемент нужного массива.

### Примеры

стандартный ввод	стандартный вывод
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	

## Задача D. СНМ

Имя входного файла: стандартный ввод  
Имя выходного файла: стандартный вывод  
Ограничение по времени: 2 секунды  
Ограничение по памяти: 256 мегабайт

Ваша задача — реализовать **Persistent Disjoint-Set-Union**. Что это значит?

Про **Disjoint-Set-Union**:

Изначально у вас есть  $n$  элементов. Нужно научиться отвечать на 2 типа запросов.

- $+ a b$  — объединить множества, в которых лежат вершины  $a$  и  $b$
- $? a b$  — сказать, лежат ли вершины  $a$  и  $b$  сейчас в одном множестве

Про **Persistent**:

Теперь у нас будет несколько копий (версий) структуры данных **Disjoint-Set-Union**.

Запросы будут выглядеть так:

- $+ i a b$  — запрос к  $i$ -й структуре, объединить множества, в которых лежат вершины  $a$  и  $b$ . При этом  $i$ -я структура остается не измененной, создается новая версия, ей присваивается новый номер (какой? читайте дальше)
- $? i a b$  — запрос к  $i$ -й структуре, сказать, лежат ли вершины  $a$  и  $b$  сейчас в одном множестве

### Формат входных данных

На первой строке 2 числа  $N$  ( $1 \leq N \leq 10^5$ ) и  $K$  ( $0 \leq K \leq 10^5$ ) — число элементов и число запросов. Изначально все элементы находятся в различных множествах. Эта начальная копия (версия) структуры имеет номер 0.

Далее следуют  $K$  строк, на каждой описание очередного запроса. Формат запросов описан выше. Запросы нумеруются целыми числами от 1 до  $K$ .

Пусть  $j$ -й из  $K$  запросов имеет вид « $+ i a b$ ». Тогда новая версия получит номер  $j$ . Запросы вида « $? i a b$ » не порождают новых структур.

### Формат выходных данных

Для каждого запроса вида  $? i a b$  на отдельной строке нужно вывести YES или NO.

### Примеры

стандартный ввод	стандартный вывод
4 7	NO
+ 0 1 2	YES
? 0 1 2	YES
? 1 1 2	YES
+ 1 2 3	NO
? 4 3 1	
? 0 4 4	
? 4 1 4	

## Задача Е. Различные числа online

Имя входного файла: стандартный ввод  
Имя выходного файла: стандартный вывод  
Ограничение по времени: 7 секунд  
Ограничение по памяти: 512 мегабайт

Сколько различных чисел на отрезке массива?

### Формат входных данных

На первой строке длина массива  $n$  ( $1 \leq n \leq 300\,000$ ). На второй строке  $n$  целых чисел от 0 до  $10^9-1$ . На третьей строке количество запросов  $q$  ( $1 \leq q \leq 300\,000$ ). Следующие  $q$  строк содержат описание запросов, по одному на строке. Каждый запрос задаётся парой целых чисел  $l, r$  ( $1 \leq l \leq r \leq n$ ).

### Формат выходных данных

Выведите ответы на запросы по одному в строке.

### Примеры

стандартный ввод	стандартный вывод
5	3
1 1 2 1 3	2
3	3
1 5	
2 4	
3 5	

### Замечание

В этой задаче необходимо отвечать на запросы **online**, поэтому задача настроена как интерактивная. После вывода ответа на запрос, вы должны сбросить буфер вывода до начала ввода следующего запроса.

## Задача F. $K$ -я порядковая статистика на отрезке

Имя входного файла: kth.in  
Имя выходного файла: kth.out  
Ограничение по времени: 6 секунд  
Ограничение по памяти: 512 мегабайт

Дан массив из  $N$  неотрицательных чисел, строго меньших  $10^9$ . Вам необходимо ответить на несколько запросов о величине  $k$ -й порядковой статистики на отрезке  $[l, r]$ .

### Формат входных данных

Первая строка содержит число  $N$  ( $1 \leq N \leq 450\,000$ ) — размер массива.

Вторая строка может быть использована для генерации  $a_i$  — начальных значений элементов массива. Она содержит три числа  $a_1, l$  и  $m$  ( $0 \leq a_1, l, m < 10^9$ ); для  $i$  от 2 до  $N$

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

В частности,  $0 \leq a_i < 10^9$ .

Третья строка содержит одно целое число  $B$  ( $1 \leq B \leq 1000$ ) — количество групп запросов.

Следующие  $B$  строк описывают одну группу запросов. Каждая группа запросов описывается 10 числами. Первое число  $G$  обозначает количество запросов в группе. Далее следуют числа  $x_1, l_x$  и  $m_x$ , затем  $y_1, l_y$  и  $m_y$ , затем,  $k_1, l_k$  и  $m_k$  ( $1 \leq x_1 \leq y_1 \leq N$ ,  $1 \leq k_1 \leq y_1 - x_1 + 1$ ,  $0 \leq l_x, m_x, l_y, m_y, l_k, m_k < 10^9$ ). Эти числа используются для генерации вспомогательных последовательностей  $x_g$  и  $y_g$ , а также параметров запросов  $i_g, j_g$  и  $k_g$  ( $1 \leq g \leq G$ )

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

Сгенерированные последовательности описывают запросы,  $g$ -й запрос состоит в поиске  $k_g$ -го по величине числа среди элементов отрезка  $[i_g, j_g]$ .

Суммарное количество запросов не превосходит 600 000.

### Формат выходных данных

Выведите единственное число — сумму ответов на запросы.

### Примеры

kth.in	kth.out
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	

## Задача G. Персистентная очередь

Имя входного файла: `queue.in`  
Имя выходного файла: `queue.out`  
Ограничение по времени: 2 секунды  
Ограничение по памяти: 256 мегабайта

Реализуйте персистентную очередь.

### Формат входных данных

Первая строка содержит количество действий  $n$  ( $1 \leq n \leq 200\,000$ ). В строке номер  $i + 1$  содержится описание действия  $i$ :

- $1\ t\ m$  — добавить в конец очереди номер  $t$  ( $0 \leq t < i$ ) число  $m$ ;
- $-1\ t$  — удалить из очереди номер  $t$  ( $0 \leq t < i$ ) первый элемент.

В результате действия  $i$ , описанного в строке  $i + 1$  создается очередь номер  $i$ . Изначально имеется пустая очередь с номером ноль.

Все числа во входном файле целые, и помещаются в знаковый 32-битный тип.

### Формат выходных данных

Для каждой операции удаления выведите удаленный элемент на отдельной строке.

### Примеры

<code>queue.in</code>	<code>queue.out</code>
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	

## Задача Н. Урны и шары

Имя входного файла: `balls.in`  
Имя выходного файла: `balls.out`  
Ограничение по времени: 4 секунды  
Ограничение по памяти: 256 мегабайт

Пусть у вас есть  $n$  урн, в каждой из которых лежит по одному шару. Урна с номером  $i$  содержит шарик под номером  $i$ . У вас есть специальное устройство, которое позволяет перемещать шарики. Им чрезвычайно просто пользоваться: сначала вы выбираете некоторый отрезок последовательных урн. После этого вы выбираете некоторый другой отрезок последовательных урн такой же длины, как и исходный, и затем шарики из урн первого отрезка перемещаются в соответствующие урны второго отрезка.

Дана последовательность перемещений. Установите, в какой урне окажется каждый шарик.

### Формат входных данных

Первая строка входных данных содержит два числа  $n$  и  $m$  — число урн и число перемещений, соответственно ( $1 \leq n \leq 100\,000$ ,  $1 \leq m \leq 50\,000$ ). Каждая из следующих  $m$  строк содержит три числа  $count_i$ ,  $from_i$  и  $to_i$ , которые означают одновременное перемещение всех шариков из урны  $from_i$  в урну  $to_i$ , всех шариков из урны  $from_i + 1$  в урну  $to_i + 1$ , ..., всех шариков из урны  $from_i + count_i - 1$  в урну  $to_i + count_i - 1$  ( $1 \leq count_i, from_i, to_i \leq n$ ,  $\max(from_i, to_i) + count_i \leq n + 1$ ).

### Формат выходных данных

Выведите  $n$  чисел — итоговые позиции каждого шарика.

### Примеры

<code>balls.in</code>	<code>balls.out</code>
2 3	1 1
1 1 2	
1 2 1	
1 2 1	

## Задача J. Persistent List

Имя входного файла: `stdin`  
Имя выходного файла: `stdout`  
Ограничение по времени: 3 секунды  
Ограничение по памяти: 512 мегабайт

Даны  $N$  списков. Каждый состоит из одного элемента.

Нужно научиться совершать следующие операции:

- `merge` — взять два каких-то уже существующих списка и породить новый, равный их конкатенации.
- `head` — взять какой-то уже существующий список  $L$  и породить два новых, в одном первый элемент  $L$ , во втором весь  $L$  кроме первого элемента.
- `tail` — взять какой-то уже существующий список  $L$  и породить два новых, в одном весь  $L$  кроме последнего элемента, во втором последний элемент  $L$ .

Для свежесозданных списков нужно говорить сумму элементов в них по модулю  $10^9 + 7$ .

### Формат входных данных

Число  $N$  ( $1 \leq N \leq 10^5$ ). Далее  $N$  целых чисел от 1 до  $10^9$  — элементы списков. Исходные списки имеют номера  $1, 2, \dots, N$ .

Затем число  $M$  ( $1 \leq M \leq 10^5$ ) — количество операций. Далее даны операции в следующем формате:

- `merge i j`
- `head i`
- `tail i`

Где  $i$  и  $j$  — номера уже существующих списков. Если в текущий момент имеется  $K$  списков, новый список получает номер  $K + 1$ .

Для операций `head` и `tail` считается, что сперва порождается левая часть, затем правая (см. пример). Также вам гарантируется, что никогда не будут порождаться пустые списки.

### Формат выходных данных

Для каждого нового списка нужно вывести сумму элементов по модулю  $10^9 + 7$ .

### Примеры

stdin	stdout
4	3
1 2 3 4	7
7	10
merge 1 2	3
merge 3 4	7
merge 6 5	5
head 7	2
tail 9	5
merge 2 3	2
merge 1 1	

## Задача К. Откат

Имя входного файла: `rollback.in`  
Имя выходного файла: `rollback.out`  
Ограничение по времени: 2 секунды  
Ограничение по памяти: 256 мегабайта

Сергей работает системным администратором в очень крупной компании. Естественно, в круг его обязанностей входит резервное копирование информации, хранящейся на различных серверах и «откат» к предыдущей версии в случае возникновения проблем.

В данный момент Сергей борется с проблемой недостатка места для хранения информации для восстановления. Он решил перенести часть информации на новые сервера. К сожалению, если что-то случится во время переноса, он не сможет произвести откат, поэтому процедура переноса должна быть тщательно спланирована.

На данный момент у Сергея хранятся  $n$  точек восстановления различных серверов, пронумерованных от 1 до  $n$ . Точка восстановления с номером  $i$  позволяет произвести откат для сервера  $a_i$ . Сергей решил разбить перенос на этапы, при этом на каждом этапе в случае возникновения проблем будут доступны точки восстановления с номерами  $l, l + 1, \dots, r$  для некоторых  $l$  и  $r$ .

Для того, чтобы спланировать перенос данных оптимальным образом, Сергею необходимо научиться отвечать на запросы: для заданного  $l$ , при каком минимальном  $r$  в процессе переноса будут доступны точки восстановления не менее чем  $k$  различных серверов.

Помогите Сергею.

### Формат входных данных

Первая строка входного файла содержит два целых числа  $n$  и  $m$ , разделенные пробелами — количество точек восстановления и количество серверов ( $1 \leq n, m \leq 100\,000$ ). Вторая строка содержит  $n$  целых чисел  $a_1, a_2, \dots, a_n$  — номера серверов, которым соответствуют точки восстановления ( $1 \leq a_i \leq m$ ).

Третья строка входного файла содержит  $q$  — количество запросов, которые необходимо обработать ( $1 \leq q \leq 100\,000$ ). В процессе обработки запросов необходимо поддерживать число  $p$ , исходно оно равно 0. Каждый запрос задается парой чисел  $x_i$  и  $y_i$ , используйте их для получения данных запроса следующим образом:  $l_i = ((x_i + p) \bmod n) + 1$ ,  $k_i = ((y_i + p) \bmod m) + 1$  ( $1 \leq l_i, x_i \leq n$ ,  $1 \leq k_i, y_i \leq m$ ). Пусть ответ на  $i$ -й запрос равен  $r$ . После выполнения этого запроса, следует присвоить  $p$  значение  $r$ .

### Формат выходных данных

На каждый запрос выведите одно число — искомое минимальное  $r$ , либо 0, если такого  $r$  не существует.

### Примеры

<code>rollback.in</code>	<code>rollback.out</code>
7 3	1
1 2 1 3 1 2 1	4
4	0
7 3	6
7 1	
7 1	
2 2	

## Задача L. Соединение и разъединение

Имя входного файла: `connect.in`  
Имя выходного файла: `connect.out`  
Ограничение по времени: 3 секунды  
Ограничение по памяти: 512 мегабайт

Вы когда-нибудь слышали про обход в глубину? Например, используя этот алгоритм, вы можете проверить является ли граф связным за время  $O(E)$ . Вы можете даже посчитать количество компонент связности за то же время.

А вы когда-нибудь слышали про систему непересекающихся множеств? Используя эту структуру, вы можете быстро обрабатывать запросы “Добавить ребро в граф” и “Посчитать количество компонент связности в графе”.

А вы когда-нибудь слышали о *динамической* задаче связности? В этой задаче вам необходимо обрабатывать три типа запросов:

1. Добавить ребро в граф.
2. Удалить ребро из графа.
3. Посчитать количество компонент связности в графе.

Можно считать, что граф является неориентированным. Изначально граф является пустым.

### Формат входных данных

В первой строке находятся два целых числа  $N$  и  $K$  — количество вершин и количество запросов, соответственно ( $1 \leq N \leq 300\,000$ ,  $0 \leq K \leq 300\,000$ ). Следующие  $K$  строк содержат запросы, по одному в строке. Каждый запрос имеет один из трех типов:

1.  $+ u v$ : Добавить ребро между вершинами  $u$  и  $v$ . Гарантируется, что такого ребра нет.
2.  $- u v$ : Удалить ребро между  $u$  и  $v$ . Гарантируется, что такое ребро есть.
3.  $?$ : Посчитать количество компонент связности в графе.

Вершины пронумерованы целыми числами от 1 до  $N$ . Во всех запросах  $u \neq v$ .

### Формат выходных данных

Для каждого запроса типа ‘?’, Выведите количество компонент связности в момент запроса.

### Примеры

<code>connect.in</code>	<code>connect.out</code>
5 11	5
?	1
+ 1 2	1
+ 2 3	2
+ 3 4	
+ 4 5	
+ 5 1	
?	
- 2 3	
?	
- 4 5	
?	