Задача A. Персистентный стек

Имя входного файла:	`stack.in`
Имя выходного файла:	`stack.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	64 мегабайта

Реализуйте персистентный стек.

Формат входного файла

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

- `t m` — добавить в конец стека номер t ($0 \le t < i$) число m ($0 < m \le 1000$);
- `t ∅` — удалить последний элемент стека номер t ($0 \le t < i$). Гарантируется, что стек t не пустой.

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

Все числа во входном файле целые.

Формат выходного файла

Для каждой операции удаления выведите удаленный элемент на отдельной строке.

Примеры

stack.in	stack.out
8 0 1 1 5 2 4 3 2 4 3 5 0 6 6 1 0	3 1

Задача B. Персистентная очередь

Имя входного файла:	`queue.in`
Имя выходного файла:	`queue.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	64 мегабайта

Реализуйте персистентную очередь.

Формат входного файла

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

- `1 t m` — добавить в конец очереди номер t ($0 \le t < i$) число m;
- `-1 t` — удалить из очереди номер t ($0 \le t < i$) 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 выходного файла

Для каждой операции удаления выведите удаленный элемент на отдельной строке.

Примеры

queue.in	queue.out
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

Задача C. Менеджер памяти

Имя входного файла:	`memory.in`
Имя выходного файла:	`memory.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	64 мегабайта

Одно из главных нововведений новейшей операционной системы Indows 7 — новый менеджер памяти. Он работает с массивом длины N и позволяет выполнять три самые современные операции:

- `copy(a, b, l)` — скопировать отрезок длины $[a, a+l-1]$ в $[b, b+l-1]$
- `sum(l, r)` — посчитать сумму элементов массива на отрезке $[l, r]$
- `print(l, r)` — напечатать элементы с l по r, включительно

Вы являетесь разработчиком своей операционной системы, и Вы, безусловно, не можете обойтись без инновационных технологий. Вам необходимо реализовать точно такой же менеджер памяти.

Формат входного файла

Первая строка входного файла содержит целое число N ($1 \le N \le 1\,000\,000$) — размер массива, с которым будет работать Ваш менеджер памяти.

Во второй строке содержатся четыре числа $1 \le X_1, A, B, M \le 10^9 + 10$. С помощью них можно сгенерировать исходный массив чисел $X_1, X_2, \cdots, X_N$. $X_{i+1} = (A * X_i + B) \mod M$

Следующая строка входного файла содержит целое число K ($1 \le K \le 200\,000$) — количество запросов, которые необходимо выполнить Вашему менеджеру памяти.

Далее в K строках содержится описание запросов. Запросы заданы в формате:

- `cpy` a b l — для операции `copy`
- `sum` l r — для операции `sum` ($l \le r$)
- `out` l r — для операции `print` ($l \le r$)

Гарантируется, что суммарная длина запросов `print` не превышает 3 000. Также гарантируется, что все запросы корректны.

Формат выходного файла

Для каждого запроса *sum* или *print* выведите в выходной файл на отдельной строке результат запроса.

Примеры

memory.in	memory.out
6 1 4 5 7 7 out 1 6 cpy 1 3 2 out 1 6 sum 1 4 cpy 1 2 4 out 1 6 sum 1 6	1 2 6 1 2 6 1 2 1 2 2 6 6 1 1 2 1 2 6 13

Задача D. 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 $l, l+1, \ldots, r$ 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 входного файла

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

Третья строка входного файла содержит q — количество запросов, которые необходимо обработать ($1 \le q \le 100\,000$). 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: $l_i = ((x_i + p) \bmod n) + 1$, $k_i = ((y_i + p) \bmod m) + 1$ ($1 \le l_i, x_i \le n$, $1 \le k_i, y_i \le m$). Пусть ответ на i-й запрос равен r. После выполнения этого запроса, следует присвоить p значение r.

Формат выходного файла

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

Примеры

rollback.in	rollback.out
7 3 1 2 1 3 1 2 1 4 7 3 7 1 7 1 2 2	1 4 0 6