Использовать стандартную библиотеку (std::priority_queue, java.util.PriorityQueue, std::set, java.util.TreeSet) не разрешается. Все задачи решаются с помощью СНМ или приоритетных очередей, никаких неизученных структур данных не требуется.

Задача A. Хип ли?

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

Структуру данных Heap можно реализовать на основе массива.

Для этого должно выполнятся *основное свойство Heap'a*, которое заключается в следующем. Для каждого $1 \leqslant i \leqslant n$ выполняются следующие условия:

- Если $2i \leqslant n$, то $a[i] \leqslant a[2i]$
- Если $2i + 1 \leqslant n$, то $a[i] \leqslant a[2i + 1]$

Дан массив целых чисел. Определите является ли он Heap'ом.

Формат входных данных

Первая строка входного файла содержит целое число n ($1 \leqslant n \leqslant 10^5$). Вторая строка содержит n целых чисел по модулю не превосходящих $2 \cdot 10^9$.

Формат выходных данных

Выведите «`YES`», если массив является Heap'ом и «`NO`» в противном случае.

Пример

isheap.in	isheap.out
5 1 0 1 2 0	NO
5 1 3 2 5 4	YES

Задача B. Система непересекающихся множеств

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

Реализуйте систему непересекающихся множеств. Вместе с каждым множеством храните минимальный, максимальный элемент в этом множестве и их количество.

Формат входных данных

Первая строка входного файла содержит n — количество элементов в носителе ($1 \leqslant n \leqslant 100000$). Далее операций с множеством. Операция `get` должна возвращать минимальный, максимальный элемент в соответствующем множестве, а также их количество.

Формат выходных данных

Выведите последовательно результат выполнения всех операций `get`.

Пример

dsu.in	dsu.out
5 union 1 2 get 3 get 2 union 2 3 get 2 union 1 3 get 5 union 4 5 get 5 union 4 1 get 5	3 3 1 1 2 2 1 3 3 5 5 1 4 5 2 1 5 5

Задача C. Приоритетная очередь

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

Реализуйте приоритетную очередь. Ваша очередь должна поддерживать следующие операции: добавить элемент, извлечь минимальный элемент, уменьшить элемент, добавленный во время одной из операций.

Все операции нумеруются по порядку, начиная с 1.

Формат входных данных

Входной файл содержит описание операций со очередью. В очередь помещаются и извлекаются только целые числа, не превышающие по модулю 10^9.

Формат выходных данных

Выведите последовательно результат выполнения всех операций `extract-min`. Если перед очередной операции `extract-min` очередь пуста, выведите вместо числа звездочку.

Пример

`priorityqueue.in`	`priorityqueue.out`

`push 3`
`push 4`
`push 2`
`extract-min`
`decrease-key 2 1`
`extract-min`
`extract-min`
`extract-min`
	`2`
	`1`
	`3`
	`*`

Задача D. Парковка

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

На кольцевой парковке есть n мест пронумерованых от 1 до n. Всего на парковку приезжает n машин в порядке нумерации. У i-й машины известно место p_i, которое она хочет занять. Если машина приезжает на парковку, а её место занято, то она едет далее по кругу и встаёт на первое свободное место.

Формат входных данных

В первой строке входного файла находится число n ($1 \leqslant n \leqslant 300\,000$) — размер парковки и число машин. Во второй строке записаны n чисел, i-е из которых p_i ($1 \leqslant p_i \leqslant n$) — место, которое хочет занять машина с номером i.

Формат выходных данных

Выведите n чисел: i-е число — номер парковочного места, которое было занято машиной с номером i.

Пример

parking.in	parking.out
3 2 2 2	2 3 1
3 1 1 2	1 2 3

Задача E. Yet another set merging

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

Вам задано n различных чисел от 1 до n. Изначально каждое число лежит в своем множестве. Обозначим множество, которому принадлежит элемент i как $s(i)$.

Ваша задача поддерживать три вида операций:

1. Объединить множества $s(x)$ и $s(y)$, где $1 \leqslant x, y \leqslant n$. Если $s(x)$ совпадает с $s(y)$, ничего делать не требуется.
2. Объединить множества $s(x), s(x+1), \ldots, s(y)$, где $1 \leqslant x \leqslant y \leqslant n$.
3. Ответить на вопрос, правда ли, что числа x и y лежат в одном множестве ($1 \leqslant x, y \leqslant n$).

Формат входных данных

Первая строка входных данных содержит два целых числа n и q ($1 \leqslant n \leqslant 200\,000$, $1 \leqslant q \leqslant 500\,000$) — количество чисел и количество запросов.

В последующих q строках находятся запросы. Каждый запрос имеет вид $type$ x y, где $type \in \{1, 2, 3\}$. Обратите внимание, что x может равняться y в запросе любого типа.

Формат выходных данных

На каждый запрос типа 3 выведите «`YES`» или «`NO`» (без кавычек), в зависимости от того, находятся ли числа в одном множестве.

Примеры

restructure.in	restructure.out
8 6 3 2 5 1 2 5 3 2 5 2 4 7 2 1 2 3 1 7	NO YES YES

Задача F. Разрезание графа

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

Дан неориентированный граф. Над ним в заданном порядке производят операции следующих двух типов:

- `cut` — разрезать граф, то есть удалить из него ребро;
- `ask` — проверить, лежат ли две вершины графа в одной компоненте связности.

Известно, что после выполнения всех операций типа `cut` рёбер в графе не осталось. Найдите результат выполнения каждой из операций типа `ask`.

Формат входных данных

Первая строка входного файла содержит три целых числа, разделённые пробелами — количество вершин графа n, количество рёбер m и количество операций k ($1 \leqslant n \leqslant 50\,000$, $0 \leqslant m \leqslant 100\,000$, $m \leqslant k \leqslant 150\,000$).

Следующие m строк задают рёбра графа; i-я из этих строк содержит два числа u_i и v_i ($1 \leqslant u_i, v_i \leqslant n$), разделённые пробелами — номера концов i-го ребра. Вершины нумеруются с единицы; граф не содержит петель и кратных рёбер.

Далее следуют k строк, описывающих операции. Операция типа `cut` задаётся строкой "`cut` u v" ($1 \leqslant u, v \leqslant n$), которая означает, что из графа удаляют ребро между вершинами u и v. Операция типа `ask` задаётся строкой "`ask` u v" ($1 \leqslant u, v \leqslant n$), которая означает, что необходимо узнать, лежат ли в данный момент вершины u и v в одной компоненте связности. Гарантируется, что каждое ребро графа встретится в операциях типа `cut` ровно один раз.

Формат выходных данных

Для каждой операции `ask` во входном файле выведите на отдельной строке слово "`YES`", если две указанные вершины лежат в одной компоненте связности, и "`NO`" в противном случае. Порядок ответов должен соответствовать порядку операций `ask` во входном файле.

Примеры

cutting.in	cutting.out
3 3 7 1 2 2 3 3 1 ask 3 3 cut 1 2 ask 1 2 cut 1 3 ask 2 1 cut 2 3 ask 3 1	YES YES NO NO