

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 А. 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 выполняться *основное свойство Heap'a*, которое заключается в следующем. Для каждого  $1 \leq i \leq n$  выполняются следующие условия:

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

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

### Формат входных данных

Первая строка входного файла содержит целое число  $n$  ( $1 \leq n \leq 10^5$ ). 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

<code>isheap.in</code>	<code>isheap.out</code>
5 1 0 1 2 0	NO
5 1 3 2 5 4	YES

## Задача В. 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 ( $1 \leq n \leq 300\,000$ ). 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	3 3 1
union 1 2	1 2 2
get 3	1 3 3
get 2	5 5 1
union 2 3	4 5 2
get 2	1 5 5
union 1 3	
get 5	
union 4 5	
get 5	
union 4 1	
get 5	

## Задача С. 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 операцией `extract-min` очередь пуста, выведите вместо числа звездочку.

### Пример

<code>priorityqueue.in</code>	<code>priorityqueue.out</code>
<code>push 3</code>	<code>2</code>
<code>push 4</code>	<code>1</code>
<code>push 2</code>	<code>3</code>
<code>extract-min</code>	<code>*</code>
<code>decrease-key 2 1</code>	
<code>extract-min</code>	
<code>extract-min</code>	
<code>extract-min</code>	

## 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 пронумерованных 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  $n$  ( $1 \leq n \leq 300\,000$ ) — размер парковки и число машин. Во второй строке записаны  $n$  чисел,  $i$ -е из которых  $p_i$  ( $1 \leq p_i \leq n$ ) — 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  $1 \leq x, y \leq n$ . Если  $s(x)$  совпадает с  $s(y)$ , ничего делать не требуется.
2. Объединить множества  $s(x), s(x+1), \dots, s(y)$ , где  $1 \leq x \leq y \leq n$ .
3. Ответить на вопрос, правда ли, что числа  $x$  и  $y$  лежат в одном множестве ( $1 \leq x, y \leq n$ ).

### Формат входных данных

Первая строка входных данных содержит два целых числа  $n$  и  $q$  ( $1 \leq n \leq 200\,000$ ,  $1 \leq q \leq 500\,000$ ) — 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	NO
3 2 5	YES
1 2 5	YES
3 2 5	
2 4 7	
2 1 2	
3 1 7	

## 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  $k$  ( $1 \leq n \leq 50\,000$ ,  $0 \leq m \leq 100\,000$ ,  $m \leq k \leq 150\,000$ ).

Следующие  $m$  строк задают рёбра графа;  $i$ -я из этих строк содержит два числа  $u_i$  и  $v_i$  ( $1 \leq u_i, v_i \leq n$ ), 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**  $u$   $v$ ” ( $1 \leq u, v \leq n$ ), которая означает, что из графа удаляют ребро между вершинами  $u$  и  $v$ . Операция типа **ask** задаётся строкой “**ask**  $u$   $v$ ” ( $1 \leq u, v \leq n$ ), 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	YES
1 2	YES
2 3	NO
3 1	NO
ask 3 3	
cut 1 2	
ask 1 2	
cut 1 3	
ask 2 1	
cut 2 3	
ask 3 1	

## Задача G. RMQ Наоборот

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

Рассмотрим массив  $a[1..n]$ . Пусть  $Q(i, j)$  — ответ на запрос о нахождении минимума среди чисел  $a[i], \dots, a[j]$ . Вам даны несколько запросов и ответы на них. Известно, что ответы на запросы непротиворечивы, восстановите исходный массив.

### Формат входных данных

Первая строка входного файла содержит число  $n$  — размер массива, и  $m$  — число запросов ( $1 \leq n, m \leq 300\,000$ ). Следующие  $m$  строк содержат по три целых числа  $i, j$  и  $q$ , означающих, что  $Q(i, j) = q$  ( $1 \leq i \leq j \leq n, -2^{31} \leq q \leq 2^{31} - 1$ ).

Гарантируется, что ответы на запросы непротиворечивы.

### Формат выходных данных

Выведите элементы массива. Элементами массива должны быть целые числа в интервале от  $-2^{31}$  до  $2^{31} - 1$  включительно. Если решений несколько, выведите любое. Гарантируется, что существует ответ с заданными ограничениями.

### Пример

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

## Задача Н. Двудольный граф

Имя входного файла: стандартный ввод  
Имя выходного файла: стандартный вывод  
Ограничение по времени: 1 секунда  
Ограничение по памяти: 512 мегабайт

Задан неориентированный граф на  $N$  вершинах без рёбер. Каждая вершина графа в каждый момент времени покрашена в один из цветов: 0 или 1 - и верно, что цвета вершин, между которыми есть ребро, различны.

Поступают два вида запросов:

- Заданы два элемента  $x, y$  из разных компонент связности: добавить в граф ребро  $(x, y)$ , изменив цвета вершин для соответствия условию.
- Заданы два элемента  $x, y$  из одной компоненты связности: ответить, правда ли, что их цвета различаются

### Формат входных данных

В первой строке содержится два целых числа  $N, M$ . ( $1 \leq N, M \leq 2 \cdot 10^5$ )

В последующих  $M$  строках содержится описание запросов.

- Запрос первого вида имеет вид:  $0 \ a \ b$ , означающий, что нужно объединить компоненты  $x$  и  $y$ , такие что  $x \bmod n = (a + shift) \bmod n$   $y \bmod n = (b + shift) \bmod n$
- Запрос второго типа имеет вид:  $1 \ a \ b$ , означающий, что нужно проверить различность цветов вершин  $x, y$ , таких что  $x \bmod n = (a + shift) \bmod n$   $y \bmod n = (b + shift) \bmod n$

Если ответ на этот запрос - да, то нужно выполнить присвоение  $shift = (shift + 1) \bmod n$

### Пример

стандартный ввод	стандартный вывод
3 5	NO
0 1 2	YES
0 2 3	NO
1 1 2	
1 1 3	
1 1 3	



## Задача I. Ещё одна задача про деревья

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

Есть граф из  $N$  вершин. Изначально он пустой. Нужно обработать  $M$  запросов:

1. добавить ребро из вершины  $v_1$  в вершину  $v_2$ , при этом вершины  $v_1$  и  $v_2$  находятся в разных деревьях и вершина  $v_2$  является корнем какого-то дерева.
2. по двум вершинам  $a$  и  $b$  определить, лежат ли они в одном дереве.

Решение задачи: реализуем СНМ с эвристикой сжатия путей:

```
int n, m, p[NMAX];

int find_set(int v)
{
    if (p[v] != v)
        p[v] = find_set(p[v]);
    return p[v];
}

int main()
{
    scanf ("%d%d", &n, &m);
    for (int i = 1; i <= n; i++)
        p[i] = i;
    for (int i = 1; i <= m; i++)
    {
        int x, y, z;
        scanf ("%d%d%d", &z, &x, &y);
        if (z == 1) {
            p[y] = x;
        } else {
            if (find_set(x) == find_set(y))
                printf ("YES\n");
            else
                printf ("NO\n");
        }
    }
}
```

Вам же предстоит сделать тест, на котором это решение будет работать долго. Более точно, нужно сделать тест, на котором количество вызовов функции `find_set` будет не меньше, чем  $\frac{1}{4}M \log_2 M$ .

### Формат входных данных

Входной файл содержит два целых числа  $N$  и  $M$  ( $1 \leq N \leq 10^6, 1 \leq M \leq 10^5, M \leq N$ ).

### Формат выходных данных

Выведите  $M$  строк.  $i$ -ая строка должна иметь вид  $1 \ x \ y$ , если  $i$ -ый запрос заключается в добавлении ребра из вершины  $x$  в вершину  $y$ , или  $0 \ x \ y$ , если спрашивается, лежат ли вершины  $x$  и  $y$  в одном дереве.

## Пример

стандартный ввод	стандартный вывод
2 2	1 1 2 0 1 2

## Задача J. Всем чмоки в этом чатике!

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

Сегодня Мэри, как программисту социальной сети «Телеграфчик», предстоит реализовать сложную систему управления чатами.

Задача Мэри усложняется тем, что в социальную сеть «Телеграфчик» внедрена продвинутая система шифрования «ZergRus», простая, как всё гениальное. Суть её в том, что в системе хранится одна переменная  $zerg$ , которая принимает значения от 0 (включительно) до  $p = 10^6 + 3$  (исключая  $p$ ) и меняется в зависимости от событий в системе.

В социальной сети всего  $n$  пользователей ( $1 \leq n \leq 10^5$ ). В начале дня каждый пользователь оказывается в своём собственном чате, в котором больше никого нет. Переменная  $zerg$  в начале дня устанавливается равной 0.

В течение дня происходят события типов:

1. Участник с номером  $(i + zerg) \bmod n$  посылает сообщение всем участникам, сидящим с ним в чате (в том числе и себе самому), при этом переменная  $zerg$  заменяется на  $(30 \cdot zerg + 239) \bmod p$ .
2. Происходит слияние чатов, в которых сидят участники с номерами  $(i + zerg) \bmod n$  и  $(j + zerg) \bmod n$ . Если участники и так сидели в одном чате, то ничего не происходит. Если в разных, то чаты объединяются, а переменной  $zerg$  присваивается значение  $(13 \cdot zerg + 11) \bmod p$ .
3. Участник с номером  $(i + zerg) \bmod n$  хочет узнать, сколько сообщений он не прочитал, и прочитать их. Если участник прочитал  $q$  новых сообщений, то переменной  $zerg$  присваивается значение  $(100\,500 \cdot zerg + q) \bmod p$ .

Вы поможете Мэри реализовать систему, обрабатывающую эти события?

### Формат входных данных

В первой строке входного файла записаны натуральные числа  $n$  ( $1 \leq n \leq 10^5$ ) — число пользователей социальной сети. и  $m$  ( $1 \leq m \leq 3 \cdot 10^5$ ) — число событий, произошедших за день. В следующих  $m$  строках содержится описание событий. Первое целое число в строке означает тип события  $t$  ( $1 \leq t \leq 3$ ). Если  $t = 1$ , далее следует число  $i$  ( $0 \leq i < n$ ), по которому можно вычислить, какой участник послал сообщение. Если  $t = 2$ , далее следуют числа  $i$  и  $j$  ( $0 \leq i, j < n$ ), по которым можно вычислить номера участников, чаты с которыми должны объединиться. Если  $t = 3$ , далее следует число  $i$  ( $0 \leq i < n$ ), по которому можно вычислить номер участника, желающего узнать, сколько у него сообщений, и прочитать их.

### Формат выходных данных

Для каждого события типа 3 нужно вывести число непрочитанных сообщений у участника.

### Примеры

стандартный ввод	стандартный вывод	Пояснение
4 10	1	4 10
1 0	1	1 0
1 2	2	1 1
1 1		1 2
1 2		1 3
3 1		3 0
2 1 2		2 0 1
1 3		1 1
3 3		3 0
2 3 2		2 2 1
3 2		3 1

## Задача К. Биномиальная куча

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

Реализуйте биномиальную кучу.

### Формат входных данных

В первой строке содержится два целых числа:  $N$  — общее количество куч и  $M$  — количество операций ( $1 \leq N \leq 1000, 1 \leq M \leq 1\,000\,000$ ). Изначально все кучи пусты.

Требуется поддерживать следующие операции:

- $0\ v\ a$  — добавить элемент со значением  $v$  в кучу с номером  $a$ . Вновь добавленный элемент имеет уникальный индекс равный порядковому номеру соответствующей операции добавления. Нумерация начинается с единицы.
- $1\ a\ b$  — переложить все элементы из кучи с номером  $a$  в кучу с номером  $b$ . После этой операции куча  $a$  становится пустой.
- $2\ i$  — удалить элемент с индексом  $i$ .
- $3\ i\ v$  — присвоить элементу с индексом  $i$  значение  $v$ . Гарантируется, что элемент существует.
- $4\ a$  — вывести на отдельной строке значение минимального элемента в куче с номером  $a$ . Гарантируется, что куча не пуста.
- $5\ a$  — удалить минимальный элемент из кучи с номером  $a$ . Если таковых несколько, то выбирается элемент с минимальным индексом. Гарантируется, что куча не пуста.

### Формат выходных данных

Для каждой операции поиска минимального элемента выведите единственное число: значение искомого элемента.

### Примеры

стандартный ввод	стандартный вывод
3 19	10
0 1 10	5
4 1	7
0 2 5	7
0 2 7	10
4 2	3
3 2 20	10
4 2	8
1 2 1	
4 1	
5 1	
4 1	
3 2 3	
4 1	
2 2	
4 1	
0 1 9	
1 1 3	
0 3 8	
4 3	

## Задача L. Снеговики

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

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 Учитель 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  $m$  ( $0 < m \leq 1000$ );
- $t\ \emptyset$  — 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

стандартный ввод	стандартный вывод
8 0 1 1 5 2 4 3 2 4 3 5 0 6 6 1 0	74

## Задача М. Персистентная приоритетная очередь

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

Требуется реализовать структуру данных, которая хранит мультимножество и умеет изменять любую свою предыдущую версию, выполняя одну из этих операций:

1. Заданы  $v$  и  $x$ , требуется добавить в множество  $v$  элемент со значением  $x$ , после чего вывести минимальный элемент в получившемся множестве.
2. Заданы  $v$  и  $u$ , требуется объединить множества с номерами  $v$  и  $u$ , после чего вывести минимальный элемент в получившемся множестве.
3. Задано  $v$ , требуется вывести минимальный элемент в множестве  $v$ , после чего удалить минимальный элемент из множества  $v$ . Если множество пустое, то вывести, что множество пустое, и создать новое пустое множество.

Изначально есть одно пустое множество с номером 0. После операции с номером  $i$  множество, получаемое во время этой операции, получает номер  $i$ .

### Формат входных данных

Первая строка содержит число  $n$  — количество операций для выполнения.

От вас потребуется отвечать на запросы в онлайн, при этом поддерживая переменную  $s$ . Она изначально равна нулю. После каждой операции, она пересчитывается следующим образом через предыдущее значение: если ответ на запрос равен  $x$ , то  $s = (s_{old} + x) \bmod 239017$ . Если же ответом на запрос является слово **empty**, то  $s$  не изменяется.

В следующих  $n$  строках заданы запросы.

Запросы первого типа описываются строкой **1 a b**, где  $a$  и  $b$  — неотрицательные целые числа, которые описывают  $v$  и  $x$  для соответствующего запроса, как  $v = (a + s) \bmod i$  и  $x = (b + 17s) \bmod (10^9 + 1)$ , где  $i$  — номер соответствующего запроса.

Запросы второго типа описываются строкой **2 a b**, где  $a$  и  $b$  — неотрицательные целые числа, которые описывают  $v$  и  $u$  для соответствующего запроса, как  $v = (a + s) \bmod i$  и  $u = (b + 13s) \bmod i$ , где  $i$  — номер соответствующего запроса.

Запросы третьего типа описываются строкой **3 a**, где  $a$  — неотрицательное целое число, которые описывает  $v$  для соответствующего запроса, как  $v = (a + s) \bmod i$ , где  $i$  — номер соответствующего запроса.

Число запросов не превышает 200 000. Гарантируется, что мощность любого созданного мультимножества не превышает  $2^{63}$ .

### Формат выходных данных

Требуется вывести ровно  $n$  строк, в каждой строке должно находиться неотрицательное целое число либо слово **empty**.

Для запросов первого и второго типа требуется вывести значение минимального элемента в только что созданном множестве, либо слово **empty**, если множество пустое.

Для запросов третьего типа требуется вывести минимальный элемент в множестве, либо слово **empty**, если множество пустое.

## Пример

стандартный ввод	стандартный вывод	Пояснение
9	2	9
1 0 2	3	1 0 2
1 0 999999970	2	1 0 3
2 2 0	2	2 1 2
3 0	2	3 3
2 4 4	2	2 3 1
3 0	2	3 5
3 0	3	3 6
3 0	empty	3 7
3 8		3 8