

## Задача А. Разрезанные таблицы

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

Дан массив из  $n$  чисел. Требуется написать программу, которая будет отвечать на запросы следующего вида: найти минимум на отрезке между  $u$  и  $v$  включительно.

### Формат входных данных

В первой строке входного файла даны три натуральных числа  $n$ ,  $m$  ( $1 \leq n \leq 10^5$ ,  $1 \leq m \leq 10^7$ ) и  $a_1$  ( $0 \leq a_1 < 16\,714\,589$ ) — количество элементов в массиве, количество запросов и первый элемент массива соответственно. Вторая строка содержит два натуральных числа  $u_1$  и  $v_1$  ( $1 \leq u_1, v_1 \leq n$ ) — первый запрос.

Элементы  $a_2, a_3, \dots, a_n$  задаются следующей формулой:

$$a_{i+1} = (23 \cdot a_i + 21563) \bmod 16714589.$$

Например, при  $n = 10$ ,  $a_1 = 12345$  получается следующий массив:  $a = (12345, 305498, 7048017, 11694653, 1565158, 2591019, 9471233, 570265, 13137658, 1325095)$ .

Запросы генерируются следующим образом:

$$\begin{aligned}u_{i+1} &= ((17 \cdot u_i + 751 + ans_i + 2i) \bmod n) + 1, \\v_{i+1} &= ((13 \cdot v_i + 593 + ans_i + 5i) \bmod n) + 1,\end{aligned}$$

где  $ans_i$  — ответ на запрос номер  $i$ .

Обратите внимание, что  $u_i$  может быть больше, чем  $v_i$ .

### Формат выходных данных

В выходной файл выведите  $u_m$ ,  $v_m$  и  $ans_m$  (последний запрос и ответ на него).

### Примеры

<code>sparse.in</code>	<code>sparse.out</code>
10 8 12345 3 9	5 3 1565158

## Задача В. Декартово дерево

Имя входного файла: `tree.in`  
Имя выходного файла: `tree.out`  
Ограничение по времени: 1 секунда  
Ограничение по памяти: 256 мегабайт

Вам даны пары чисел  $(a_i, b_i)$ . Необходимо построить декартово дерево, такое что  $i$ -я вершина имеет ключи  $(a_i, b_i)$ , вершины с ключом  $a_i$  образуют бинарное дерево поиска, а вершины с ключом  $b_i$  образуют кучу.

### Формат входных данных

В первой строке записано число  $N$  — количество пар. Далее следует  $N$  ( $1 \leq N \leq 50\,000$ ) пар  $(a_i, b_i)$ . Для всех пар  $|a_i|, |b_i| \leq 30\,000$ .  $a_i \neq a_j$  и  $b_i \neq b_j$  для всех  $i \neq j$ .

### Формат выходных данных

Если декартово дерево с таким набором ключей построить возможно, выведите в первой строке «YES», в противном случае выведите «NO». В случае ответа «YES» выведите  $N$  строк, каждая из которых должна описывать вершину. Описание вершины состоит из трёх чисел: номера предка, номера левого сына и номера правого сына. Если у вершины отсутствует предок или какой либо из сыновей, выведите на его месте число 0.

Если подходящих деревьев несколько, выведите любое.

### Пример

tree.in	tree.out
7	YES
5 4	2 3 6
2 2	0 5 1
3 9	1 0 7
0 5	5 0 0
1 3	2 4 0
6 6	1 0 0
4 11	3 0 0

## Задача С. Двоичное дерево поиска 1

Имя входного файла: `bst1.in`  
Имя выходного файла: `bst1.out`  
Ограничение по времени: 2 секунды  
Ограничение по памяти: 256 мегабайт

Реализуйте сбалансированное двоичное дерево поиска. Внимание! Решать задачу с использованием `set` из STL запрещено, однако рекомендуется стрессить ваше решение с ним для поиска багов

### Формат входных данных

Входной файл содержит описание операций с деревом, их количество не превышает 100000. В каждой строке находится одна из следующих операций:

- `insert x` — добавить в дерево ключ  $x$ . Если ключ  $x$  уже в дереве, то ничего делать не надо.
- `delete x` — удалить из дерева ключ  $x$ . Если ключа  $x$  в дереве нет, то ничего делать не надо.
- `exists x` — если ключ  $x$  есть в дереве, выведите «`true`», иначе «`false`»

Все числа во входном файле целые и по модулю не превышают  $10^9$ .

### Формат выходных данных

Выведите последовательно результат выполнения всех операций `exists`. Следуйте формату выходного файла из примера.

### Примеры

<code>bst1.in</code>	<code>bst1.out</code>
<code>insert 2</code>	<code>true</code>
<code>insert 5</code>	<code>false</code>
<code>insert 3</code>	
<code>exists 2</code>	
<code>exists 4</code>	
<code>delete 5</code>	

## Задача D. Двоичное дерево поиска 2

Имя входного файла: `bst2.in`  
Имя выходного файла: `bst2.out`  
Ограничение по времени: 2 секунды  
Ограничение по памяти: 256 мегабайт

Реализуйте сбалансированное двоичное дерево поиска.

### Формат входных данных

Входной файл содержит описание операций с деревом, их количество не превышает 100000. Формат операций смотрите в предыдущей задаче. В каждой строке находится одна из следующих операций:

- `insert x` — добавить в дерево ключ  $x$ .
- `delete x` — удалить из дерева ключ  $x$ . Если ключа  $x$  в дереве нет, то ничего делать не надо.
- `exists x` — если ключ  $x$  есть в дереве, выведите «`true`», иначе «`false`»
- `next x` — выведите минимальный элемент в дереве, строго больший  $x$ , или «`none`», если такого нет.
- `prev x` — выведите максимальный элемент в дереве, строго меньший  $x$ , или «`none`», если такого нет.
- `kth k` — выведите  $k$ -ый по величине элемент (нумерация с единицы). Если такого не существует, то выведите «`none`».

Все числа во входном файле целые и по модулю не превышают  $10^9$ .

### Формат выходных данных

Выведите последовательно результат выполнения всех операций `exists`, `next`, `prev`, `kth`. Следуйте формату выходного файла из примера.

### Примеры

bst2.in	bst2.out
<code>insert 2</code>	<code>true</code>
<code>insert 5</code>	<code>false</code>
<code>insert 3</code>	<code>5</code>
<code>exists 2</code>	<code>3</code>
<code>exists 4</code>	<code>none</code>
<code>next 4</code>	<code>3</code>
<code>prev 4</code>	<code>2</code>
<code>delete 5</code>	<code>none</code>
<code>next 4</code>	
<code>prev 4</code>	
<code>kth 1</code>	
<code>kth 3</code>	