

## 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>    |                    |