Задача 2. Битовый массив

Источник:	базовая*
Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда*
Ограничение по памяти:	**8 мегабайт**

Память современных компьютеров можно представить в виде очень длинного массива битов, в котором каждый бит может принимать значения 0 или 1. Однако возможность обращаться к конкретному биту по его номеру отсутствует: минимальная адресуемая единица памяти — это байт, обычно состоящий из 8 битов. Чтобы работать с отдельными битами памяти, необходимо обращаться к байтам (или словам), которые их содержат, и применять различные битовые операции.

В данной задаче нужно реализовать структуру данных “битовый массив” / “bitset”. Это массив, элементы которого принимают лишь значения 0 и 1. Эти элементы можно легко сохранить в массиве байтов, храня одно значение в каждом байте. Однако если сохранять по 8 элементов в байте, то снижается потребление памяти и появляется возможность ускорить некоторые операции — именно это и требуется сделать в данной задаче.

Нужно реализовать набор функцию со следующей сигнатурой:

```
//какой-нибудь целочисленный тип (желательно беззнаковый)
typedef ??? bitword;

//инициализирует массив битов длины num, устанавливая все его биты в ноль
void bitsetZero(bitword *arr, int num);
//возвращает значение idx-ого бита (0 или 1)
int bitsetGet(const bitword *arr, int idx);
//устанавливает значение idx-ого бита в newval (которое равно 0 или 1)
void bitsetSet(bitword *arr, int idx, int newval);
//возвращает 1, если среди битов с номерами k
//для left <= k < right есть единичный, и 0 иначе
int bitsetAny(const bitword *arr, int left, int right);
```

Массив слов `arr` управляется вызывающим, и может указывать, к примеру, на глобальный массив достаточного размера. Вы можете самостоятельно выбрать базовый тип для слова `bitword`. Однобайтовый тип подойдёт, однако для лучшей производительности рекомендуется использовать слова большего размера. Функция `bitsetAny` определяет, есть ли в заданном окне/отрезке битов хотя бы один бит, равный единице. При реализации этой функции **необходимо** быстро обрабатывать окна большого размера: для этого нужно целиком перебирать слова, полностью попадающие внутрь окна, не перебирая отдельные биты.

При помощи реализованной структуры нужно решить тестовую задачу.

Формат входных данных

В первой строке записано целое число N — количество операций, которые нужно обработать ($1 \leqslant N \leqslant 2 \cdot 10^5$). В каждой из следующих N строк описывается одна операция над битовым массивом.

Описание операции начинается с целого числа t, обозначающего тип операции. Если $t = 0$, то это операция `bitsetZero`, и вторым целых числом в строке указан размер массива `num`. Если $t = 1$, то это операция `bitsetGet`, и в строке также записано целое число `idx` — номер бита. Значение этого бита нужно выдать в выходной файл. Если $t = 2$, то это операция

`bitsetSet`, и в строке также содержатся целые числа `idx` и `newval`. Здесь `idx` — номер бита, который нужно изменить, а `newval` — новое значение, которое надо записать (0 или 1). Если $t = 3$, то это операция `bitsetAny`, и указано ещё два параметра `left` и `right` — отрезок, на котором нужно искать единичные биты. Если единичный бит на отрезке есть, то надо вывести `some`, а если нет — то `none`.

Размер массива `num` больше нуля и не превышает $2 \cdot 10^7$. Гарантируется, что после операции `bitsetZero` с параметром `num` все последующие операции обращаются только к битам с номерами от 0 до `num`-1 включительно — как минимум до следующего вызова `bitsetZero`.

Сумма значений `num` по всем операциям `bitsetZero` не превышает 10^{10}. Сумма длин отрезков (`right-left`) по всем операциям `bitsetAny` не превышает 10^{10}.

Формат выходных данных

Для каждой операции `bitsetGet` или `bitsetAny` нужно вывести ответ в отдельной строке.

Пример

input.txt	output.txt
14 0 100 2 30 1 2 31 1 2 32 1 1 31 1 7 2 31 0 1 31 3 30 33 3 31 32 3 0 100 3 45 67 0 48 1 30	1 0 0 some none some none 0

Пояснение к примеру

Сначала инициализируется массив из 100 битов. Потом устанавливаются в 1 биты с номерами 30, 31, 32. Потом делается запрос на значения битов 31 и 7 — они равны 1 и 0 соответственно. Далее 31-ый бит зануляется и запрашивается его уже нулевое значение. Потом делаются запросы о том, есть ли единичные биты на отрезках $[30, 33)$, $[31, 32)$ и $[0, 100)$. Обратите внимание, что правый конец отрезка в данной задаче исключается. Наконец, выполняется пересоздание массива, теперь размера 48 битов, в результате 30-ый бит становится нулевым.