Задача 1. Удаление дубликатов 2

Источник:	базовая*
Имя входного файла:	`input.bin`
Имя выходного файла:	`output.bin`
Ограничение по времени:	1 секунда
Ограничение по памяти:	разумное

Дан массив A, в котором содержится n целых чисел. Нужно удалить из него дубликаты (т.е. повторы чисел), так чтобы в массиве каждое имеющееся в нём значение встречалось ровно один раз.

Если значение встречается в массиве несколько раз, то нужно удалить все его вхождения, кроме самого первого. Порядок оставшихся элементов должен быть сохранён.

Внимание: задачу требуется решать с помощью **хеш-таблицы**.

Формат входных данных

В первых четырёх байтах записано число n — сколько чисел в массиве ($1 \leqslant N \leqslant 10^6$). Далее записано n чисел, по четыре байта каждое. Все числа целые, по модулю не превышают 10^9.

Формат выходных данных

В первых четырёх байтах нужно вывести целое число k — сколько различных чисел в массиве A. Далее нужно вывести k этих чисел, по четыре байта каждое. Числа должны быть выведены в том порядке, в котором их первые входения идут в исходном массиве.

Пример

input.bin
0A 00 00 00 01 00 00 00 01 00 00 00 FE FF FF FF 04 00 00 00 03 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FE FF FF FF
output.bin
05 00 00 00 01 00 00 00 FE FF FF FF 04 00 00 00 03 00 00 00 00 00 00 00

Задача 2. Сумма чётных 2

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

Требуется реализовать функцию `foreach`, которая пробегает по всем элементам заданного **массива** целых чисел и вызывает указанный callback для каждого из этих элементов. Аналогичную функцию `foreach` нужно реализовать и для **связного списка**. Передаваемый callback должен поддерживать контекст.

Пример сигнатуры функции `foreach` для массива приведён ниже. Для связного списка реализуйте функцию `foreach` с аналогичной сигнатурой, чтобы можно было передавать в неё те же самые callback-функции.

```
//тип указателя на функцию, которую можно передавать как callback
typedef void (*callback)(void *ctx, int *value);
//здесь ctx -- это контекст, который передаётся в func первым аргументом
//последние два параметра задают массив, по элементам которого нужно пройтись
void arrayForeach(void *ctx, callback func, int *arr, int n);
```

Эти две функции требуется применить для решения следующей тестовой задачи: *дана последовательность целых чисел, требуется посчитать сумму всех её чётных элементов.* Запишите данную последовательность как в массив, так и в связный список, и вызовите в каждом случае `foreach`. Функция-callback, которую вы передаёте в `foreach`, не должна обращаться к глобальным переменным: все необходимые для её работы данные передавайте через контекст.

Формат входных данных

В первой строке содержится целое число N — количество элементов последовательности ($1 \leqslant N \leqslant 100$). Во второй строке записано N целых чисел через пробел — сама последовательность. Все элементы последовательности по абсолютной величине не превышают 100.

Формат выходных данных

Выведите два целых числа: сумму всех чётных элементов последовательности. Первое число должно соответствовать сумме, вычисленной с помощью `foreach` по массиву, а второе — сумме, вычисленной с помощью `foreach` по связному списку.

Пример

input.txt	output.txt
8 2 3 7 6 8 3 1 2	18 18

Задача 3. Цикличность случайных чисел

Источник:	базовая*
Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	2 секунды
Ограничение по памяти:	разумное

Как известно, у генератора псевдослучайных чисел есть внутреннее состояние, которое может принимать конечное количество различных значений. Из этого следует, что если достаточно долго генерировать псевдослучайные числа, то в какой-то момент они начнут повторяться. В данной задаче нужно найти, с какого момента начнётся повторение у заданного квадратичного конгруентного генератора.

Квадратичный конгруентный генератор определяется четырьмя целочисленными параметрами a, b, c и $M \geqslant 2$. Его состояние представляется целым числом `state`, которое всегда находится в диапазоне от 0 до $M - 1$ включительно. Функция перехода для этого генератора выглядит так:

```
uint64_t func(uint64_t s) {
    return (s*s*a + s*b + c) % M;
}
```

Изначально, состояние генератора `state` равно единице. Далее каждый раз, когда пользователь запрашивает новое случайное число:

1. Пользователю выдаётся текущее значение `state` в качестве случайного числа.
2. К состоянию применяется функция перехода: `state = func(state)`;

Обозначим последовательность случайных чисел, которую выдаёт генератор, через $x_0, x_1, x_2, x_3, \ldots$. Нетрудно заметить, что всегда $x_0 = 1$. Будем говорить, что в этой последовательности циклически повторяется отрезок от l до r, если $x_{l+i} = x_{r+i}$ для любого $i \geqslant 0$.

Даны параметры генератора, нужно найти отрезок от l до r, который циклически повторяется. Поскольку вариантов выбора отрезка много, требуется найти такой, у которого число r минимально возможное.

Формат входных данных

В первой строке записано целое число M — модуль генератора ($2 \leqslant M \leqslant 10^{12}$). Во второй строке записано три целых числа a, b, c — параметры генератора ($0 \leqslant a, b, c \leqslant 10^9$).

Обратите внимание, что при указанных ограничениях в функции перехода `func` может происходить беззнаковое 64-битное переполнение. Это нормально, так и должно быть.

Формат выходных данных

Выведите два целых числа l и r через пробел — отрезок, которые циклически повторяется. Среди всех возможных вариантов нужно выбрать тот, в котором число r минимальное.

Гарантируется, что в ответе $r \leqslant 2 \cdot 10^6$.

Внимание: для обнаружения совпадений нужно использовать **хеш-таблицу**.

Примеры

input.txt	output.txt
11 1 4 5	1 4
999999999999 1 0 7	977966 1389969

Задача 4. lower bound

Источник:	основная*
Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	2 секунды
Ограничение по памяти:	разумное

Функция `bsearch` из стандартной библиотеки языка C обладает большим недостатком: если в массиве нет того элемента, который мы ищем, то функция не возвращает абсолютно никакой информации. Это означает, что в некоторых случаях она бесполезна, например при решении задачи «Поиск ближайшего».

Гораздо полезнее функция `lower_bound` из стандартной библиотеки языка C++. Эта функция запускается для отсортированного массива A и элемента X, и возвращает номер первого элемента в массиве, который больше или равен X. Если такого элемента нет, то функция возвращает длину массива N. Иными словами, функция возвращает, сколько элементов в массиве A строго меньше заданного X.

В данной задаче вам предлагается реализовать `lower_bound` на языке C аналогично тому, как реализована функция `bsearch`. Это означает, что должны выполняться следующие требования:

1. Функция должна быть применима к массиву элементов любого типа. Это означает, что она должна принимать размер одного элемента в байтах и нетипизированные указатели, аналогично `qsort` и `bsearch`.
2. Функция должна поддерживать задание критерия сравнения для элементов. Это означает, что она должна принимать компаратор в виде указателя на функцию (можно без контекста).
3. Функция должна брать информацию только из своих параметров/аргументов. Иными словами, внутри неё нельзя обращаться к глобальным и статическим переменным.

Данную функцию нужно применить к двум заданным массивам: один состоит из целых чисел, а другой — из строк. Обратите внимание, что изначально массивы не отсортированы — вам следует применить к ним `qsort`.

Формат входных данных

Входные данные задаются в таком порядке: сначала массив целых чисел, затем массив строк, затем запросы для массива целых чисел, и, наконец, запросы для массива строк.

В первом блоке записано одно целое число N_1 — длина массива целых чисел ($1 \leqslant N_1 \leqslant 10^5$). Далее записаны элементы этого массива: N_1 целых чисел, каждое по абсолютной величине не превышает 10^{15}.

Во втором блоке записано одно целое число N_2 — длина массива строк ($1 \leqslant N_2 \leqslant 10^5$). Далее записаны элементы этого массива: N_2 непустых строк из маленьких букв латинского алфавита, длиной не более 31 символа каждая.

В третьем блоке записано целое число Q_1 — количество запросов для массива целых чисел ($1 \leqslant Q_1 \leqslant 10^5$). В остальных Q_1 строках записаны целые числа, определяющие запросы на поиск, каждое число не превышает 10^{15} по абсолютной величине.

В четвёртом блоке записано целое число Q_2 — количество запросов для массива строк ($1 \leqslant Q_2 \leqslant 10^5$). В остальных Q_2 строках записаны строки-запросы, состоящие из маленьких букв латинского алфавита, длиной от 1 до 31 символа.

Формат выходных данных

В выходные данные нужно вывести сначала результаты применения `lower_bound` для запросов на массиве целых чисел, а затем результаты применения для запросов на массиве строк. Первый блок результатов должен содержать Q_1 целых чисел в диапазоне от 0 до N_1, каждое число в отдельной строке. Второй блок результатов должен содержать Q_2 целых чисел в диапазоне от 0 до N_2, каждое число в отдельной строке.

Пример

input.txt	output.txt
5 -1000000000 3000000000 5000000000 -1000000000 3000000000 5 a hello a ba a 8 3000000000 2999999999 3000000001 0 5000000000 4000000000 7000000000 -7000000000 3 a b hi	2 2 4 2 4 4 5 0 0 3 5

Пояснение к примеру

Массив целых чисел в отсортированном виде выглядит как: $-G$, $-G$, $3G$, $3G$, $5G$ (для краткости обозначим $G = 10^9$). Первый запрос в точности равен $3G$, он впервые встречается под индексом 2 в массиве. Второй запрос чуть меньше $3G$, и в массиве отсутствует, так что нужно вернуть индекс первого элемента больше него, а это 2. Третий запрос чуть больше $3G$, и в массиве также отсутствует. В массиве всего 4 числа меньше запрашиваемого, так что ответ равен 4. Последние два запроса показывают, что нужно возвращать, когда запрашиваемое число больше или меньше всего массива.

Массив строк в отсортированном виде выглядит так: `a`, `a`, `a`, `ba`, `hello`. Для запроса `a` ответ равен 0, т.к. элементов меньше в массиве нет. Для запроса `b` ответ равен 3, т.к. все три строки `a` меньше него, а строка `ba` больше него. Запрос `hi` больше всего массива, так что ответ равен длине массива.

Задача 5. Лавинный эффект

Источник:	основная
Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	2 секунды
Ограничение по памяти:	разумное

Для изучения качества хеш-функции принято рассматривать такое явление, как «лавинный эффект» (avalanche effect). Грубо говоря, он показывает, на какие биты хеша влияет каждый бит ключа. Считается, что если хеш-функция хорошая, то при переключении любого бита в ключе на противоположный в выходном хеше должна поменяться примерно половина битов. Например, хорошее лавинное поведение показывает хеш-функция Дженкинса. В этой задаче предлагается вычислить лавинный эффект для заданной хеш-функции.

Хеш-функция вычисляется следующим кодом на языке C:

```
uint64_t A, B, M, R, S;
uint32_t hashFunc(uint32_t x) {
    return (((A * x + B) % M) % R) / S;
}
```

Как видно, эта функция принимает 32-битные беззнаковые ключи, и выдаёт 32-битные беззнаковые хеши. **Осторожно:** знаковость и битность всех переменных в этом коде имеет значение и должна быть именно такой, как написано!

Таблица лавинного эффекта для этой функции имеет размер 32×32, то есть в ней 32 строки и 32 столбца. В i-ой строке в j-ом столбце записано число, обозначаемое p_{ij}. Число p_{ij} равно вероятности того, что изменение i-ого бита в случайном ключе приведёт к изменению j-ого бита в его хеше. При этом считается, что при выборе случайного ключа все 32-битные беззнаковые ключи равновероятны.

Формат входных данных

В единственной строке записано пять неотрицательных целых чисел: A, B, M, R, S — параметры хеш-функции. Обратите внимание, что все эти числа 64-битные и не превышают 10^{18}. Чтобы избежать деления на ноль, параметры M, R, S гарантированно ненулевые.

Формат выходных данных

Выведите таблицу лавинного эффекта как 32 строки по 32 значения в каждой. Для каждой ячейки таблицы выведите число p_{ij} в процентах: для этого нужно умножить p_{ij} на 100 и округлить до целого.

Все выведенные числа должны быть целыми. Поскольку за короткое время не получится посчитать p_{ij} с идеальной точностью, ваше решение может немного ошибиться: ваш ответ будет засчитан в том и только в том случае, если каждое число отличается от правильного не более чем на 1 (то есть ошибка в каждой вероятности должна быть не более процента).

Примеры

input.txt	output.txt
2654435769 0 4294967296 4294967296 4096	нормалёк =)
2654435769 0 1048576 1000000000 1	фу, неее =(
938572893 2139875776 1000000007 1048576 1	отлично!
15642 322777666 10000000 1000000000 1	как-то не очень...

Пояснение к примеру

В примерах входные данные не всегда умещаются в одну строку. Кроме того, выходные данные не указаны, так как они сильно большие и не входят в текст. Набор выходных данных для всех четырёх тестов можно скачать по ссылке.

По поводу хеш-функций из примеров можно сказать следующее:

1. В первом примере задана правильная хеш-функция Кнута, выдающая 20-битный хеш:
$$f(x) = \left\lfloor \frac{(A \cdot x) \mod 2^{32}}{2^{12}} \right\rfloor$$
Как видно, лавинный эффект очень хороший: каждый бит хеша зависит хотя бы от 10-12 битов ключа.
Единственная проблема — старшие биты ключа не влияют на младшие. Если кто-то решит хешировать числа, которые все делятся на 65536, то младшие 5 битов хеша не будут варьироваться вообще. С другой стороны, таких 32-битных чисел всего 65536, и может быть в хеш-таблице они разместятся без проблем.

2. Второй пример показывает неправильную реализацию хеш-функции Кнута, когда в хеш выбираются младшие биты вместо старших:
$$f(x) = (A \cdot x) \mod 2^{20}$$
Как видно, у этой функции очень плохой лавинный эффект: на каждый бит хеша влияет намного меньше битов ключа, чем у правильной хеш-функции Кнута. В целом, эта хеш-функция показывает, почему нужно быть осторожным с модулем, являющимся степенью двойки.

3. Третий пример показывает правильный вариант линейной хеш-функции с 20 выходными битами:
$$f(x) = [(A \cdot x + B) \mod P] \mod 2^{20}$$
Здесь число P равно $10^9 + 7$ и является простым, а коэффициенты A и B выбраны произвольно. Заметьте, что перед взятием остатка от деления на 2^{20} предварительно берётся остаток от деления на простое P — это важно.
У этой функции очень хорошее лавинное поведение: вся таблица заполнена. Хотя есть ячейки, в которых зависимость слабая (стоит почти ноль или почти 100), однако в общем каждый бит хеша зависит от подавляющего большинства битов ключа.

4. В последнем примере применяется такая же хеш-функция, но убрано взятие остатка от деления на простое число, отчего лавинный эффект резко ухудшается.

Задача 6. Сравнение асимптотик

Источник:	основная
Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	2 секунды
Ограничение по памяти:	разумное

В курсе всё больше и больше делается акцент на улучшение асимптотического времени работы различных операций. При этом важно понимать, при каких изменениях асимптотическое время работы становится лучше (т.е. быстрее), а при каких хуже. В данной задаче предлагается реализовать сравнение для наиболее часто встречающихся асимптотик.

В данной задаче асимптотическое время работы задаётся как функция вида:

$$T(N) = O(p^N \cdot N^s \cdot \log^l N)$$

Здесь $p \geqslant 1$, $s \geqslant 0$ и $l \geqslant 0$ — произвольные вещественные числа. Легко видеть, что в этот класс попадают, например, асимптотики сортировки слиянием $O(N \log N)$, бинарного поиска $O(\log N)$, перебора всех N-битных чисел $O(2^N N)$.

Формат входных данных

В первой строке входного файла записано число Q — сколько тестовых случаев нужно обработать ($1 \leqslant Q \leqslant 10^5$). Далее идёт $2Q$ строк, каждая пара строк описиывает один тестовый случай, то есть две асимптотики, которые надо сравнить.

Асимптотика в полном виде записывается как: `"O( p^N N^s logN^l )"` (без кавычек). В полном виде в ней три части, обязательно отделённые друг от друга и от окружающих скобок пробелом. Других пробелов нет. Части могут быть записаны в произвольном порядке.

Вместо букв `p`, `s` и `l` в описании асимтотики записаны вещественные числа, задающие соответствующие коэффициенты. Все вещественные числа записаны с не более чем тремя знаками после десятичной точки, и лежат в пределах от 0 до 10 включительно. Кроме того, для коэффициента `p` верно: $p \geqslant 1$.

Кроме того, некоторые части могут быть опущены: в таком случае в произведении этой части нет. Если опущены все три части, то асимптотика будет записана в виде `"O( 1 )"` (без кавычек). Наконец, в компонентах `N^s` и `logN^l` может быть опущена степень: в таком случае она равна единице. Если степень опущена, то в описании отсутствует как вещественное число `s` или `l`, так и символ крышки непосредственно до него.

Замечание: рекомендуется использовать `gets`, `strtok`, `sscanf` и прочие стандартные функции для чтения асимптотики.

Формат выходных данных

В выходных данных должно быть ровно Q целых чисел, по одному числу в строке. Если в запросе первая асимптотика меньше второй, число должно быть равно -1. Если первая асимптотика больше второй, то нужно вывести 1. Наконец, если они совпадают, то нужно вывести 0.

Пример

input.txt	output.txt
6 O(2^N N^3.5 logN^7.3) O(2^N N^4 logN^7.267) O(N^3.5 logN^7) O(logN^7.000 N^3.5) O(1) O(N^2) O(N^0.5) O(logN^7) O(2^N N) O(2^N) O(N logN) O(N^1.5)	-1 0 -1 1 1 -1

Задача 7. Найти коллизию

Источник:	основная
Имя входного файла:	`stdin`
Имя выходного файла:	`stdout`
Ограничение по времени:	5 секунд
Ограничение по памяти:	разумное

В данной задаче вам предлагается найти коллизию для неизвестной вам хеш-функции, то есть указать два различных ключа, на которых значение хеш-функции совпадает.

Известно, что хеш-функция принимает 32-битное беззнаковое целое число на вход (ключ) и выдаёт 32-битное беззнаковое целое число на выход (хеш). Кроме того, известно, что хеш-функция очень хорошего качества.

Вы можете вычислять хеш-функцию на любых ключах, на каких хотите. Однако всего разрешается сделать не более $2 \cdot 10^5$ вычислений.

Протокол взаимодействия

В данной задаче ваша программа будет работать не с файлами, а совместно с программой-интерактором. Ваша программа и интерактор будут запускаться одновременно, и соединяться пайпами (теми самыми пайпами, о которых упоминалось на лекции при рассмотрении очереди и кольцевого буфера). Всё, что ваша программа выводит в `stdout`, читает интерактор из своего `stdin`, а всё, что пишет интерактор на `stdout`, читает ваша программа из своего `stdin`.

Ваша программа должна печатать команды, которые интерактор будет выполнять. Есть два типа команд:

- Команда `eval`, после которой через пробел должно быть записано 32-битное беззнаковое целое число X. Эта команда предписывает интерактору вычислить значение хеш-функции от числа X. Интерактор вычислят его и записывает искомый хеш: ваша программа должна прочитать его из `stdin` как беззнаковое 32-битное целое.

- Команда `answer`, после которой через пробел должно быть записано два 32-битных беззнаковых целых числа A и B. Этой командой ваша программа должна сообщить интерактору коллизию: числа A и B должны быть различными, но их хеш должен совпадать. После этого ваша программа должна сразу же завершить исполнение, ничего никуда больше не записывая и ничего ниоткуда не читая.

Если вы сделаете больше вычислений хеш-функции, чем разрешено, или выдадите неверный ответ командой `answer`, то ваше решение получит Wrong Answer.

Поскольку задача интерактивная, требуется:

1. Не открывать никаких файлов, **не** использовать `freopen` и `fopen`.
2. Писать команды с помощью `printf` и читать ответы с помощью `scanf`.
3. После каждой команды выводить символ перевода строки и сразу после этого выполнить: `fflush(stdout);`

Если вы забудете сделать команду `fflush`, то записанные вами в `stdout` данные останутся в буфере, и никогда не попадут в пайп, а значит интерактор никогда их не получит и всё зависнет (вердикт Timeout).

Учтите, что в этой задаче все числа беззнаковые, так что писать и читать их надо с форматом `"%u"`.

Пример

stdin	stdout
2478003845 894250524 622810134 894250524	eval 1 eval 2 eval 3 eval 4 answer 2 4

Пояснение к примеру

Обратите внимание, что в примере сначала программа печатает команды в `stdout`, а уже потом на них приходят ответы от интерактора. В данном случае у ключей 2 и 4 получается одинаковый хеш, равный 894250524.

Исполняемый файл интерактора вы можете скачать по ссылке (только для Windows). Чтобы запустить его просто поиграться, нужно использовать командную строку:

```
interactor.exe input.txt output.txt
```

Чтобы запустить его вместе с вашим решением `sol.exe`, можно использовать командную строку (предварительно надо поставить Python 3):

```
python run_interactive.py sol.exe
```

Выведенные вашим решением команды будут записаны в `output.txt`.

Задача 8. Сравнение подстрок

Источник:	повышенной сложности
Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	3 секунды
Ограничение по памяти:	разумное

Дана строка S длиной в N символов, и размер блока B. В этой строке имеется ровно $(N-K+1)$ подстрок длины B (подстрока — это часть строки, которая является непрерывным отрезком). Нужно раскрасить все эти подстроки в цвета, так чтобы одинаковые подстроки имели одинаковый цвет, а разные подстроки — разный цвет.

Формат входных данных

В первой строке записано два целых числа: N — длина строки и B — длина рассматриваемых подстрок ($1 \leqslant B \leqslant N \leqslant 10^6$).

Во второй строке дана сама строка S. Её длина равна N, и она состоит только из маленьких букв латинского алфавита.

Формат выходных данных

Выведите в единственную строку $(N - B + 1)$ целых чисел через пробел: цвета всех подстрок длины B. Все цвета должны быть в диапазоне от 0 до $K-1$ включительно, где K — количество различных цветов. Цвета нужно выводить в том порядке, в котором подстроки располагаются в строке S.

Пример

input.txt
15 3 abacabadabacaba
output.txt
0 3 1 5 0 4 2 6 0 3 1 5 0

Задача 9. Цикличность случайных чисел+

Источник:	повышенной сложности*
Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда
Ограничение по памяти:	**5 мегабайт**

Предлагается решить задачу про определение цикла квадратичного конгруентного генератора, используя $O(1)$ памяти. Очевидно, в таком решении не будет никакой хеш-таблицы.

Подробнее условие можно прочитать в задаче «Цикличность случайных чисел».

Формат входных данных

В первой строке записано целое число M — модуль генератора ($2 \leqslant M \leqslant 10^{12}$). Во второй строке записано три целых числа a, b, c — параметры генератора ($0 \leqslant a, b, c \leqslant 10^9$).

Обратите внимание, что при указанных ограничениях в функции перехода `func` может происходить беззнаковое 64-битное переполнение. Это нормально, так и должно быть.

Формат выходных данных

Выведите два целых числа l и r через пробел — отрезок, которые циклически повторяется. Среди всех возможных вариантов нужно выбрать тот, в котором число r минимальное.

Гарантируется, что в ответе $r \leqslant 2 \cdot 10^6$.

Примеры

input.txt	output.txt
11 1 4 5	1 4
999999999999 1 0 7	977966 1389969