A\*. Компаратор

|  |  |
| --- | --- |
| Ограничение времени | 0.5 секунд |
| Ограничение памяти | 8Mb |
| Ввод | стандартный ввод или input.txt |
| Вывод | стандартный вывод или output.txt |

Юный программист Саша написал свою первую тестирующую систему. Он так обрадовался тому, что она скомпилировалась, что решил пригласить школьных друзей на свой собственный контест.

Но в конце тура выяснилось, что система не умеет сортировать команды в таблице результатов. Помогите Саше реализовать эту сортировку.

Команды упорядочиваются по правилам ACM:

1. по количеству решённых задач в порядке убывания;
2. при равенстве количества решённых задач – по штрафному времени в порядке возрастания;
3. при прочих равных — по номеру команды в порядке возрастания.

Используйте в этой задаче *std::vector*, *std::sort* и собственный компаратор.

Формат ввода

Первая строка содержит натуральное число *n* *(1 ≤ n ≤ 105)* — количество команд, участвующих в контесте.

В *i*-й из следующих *n* строк записано количество решенных задач *S* *(0 ≤ S ≤ 100)* и штрафное время *T* *(0 ≤ T ≤ 105)* команды с номером *i*.

Формат вывода

В выходной файл выведите *n* чисел — номера команд в отсортированном порядке.

Пример

| **Ввод**  Скопировать ввод | **Вывод**  Скопировать вывод |
| --- | --- |
| 5  3 50  5 720  1 7  0 0  8 500 | 5  2  1  3  4 |

# B\*. Зверюшки

|  |  |
| --- | --- |
| Ограничение времени | 0.5 секунд |
| Ограничение памяти | 8Mb |
| Ввод | стандартный ввод или input.txt |
| Вывод | стандартный вывод или output.txt |

Уже долгое время в Институте Мутантов, Фич Технологических и Иностранных языков разводят милых разноцветных зверюшек. Для удобства каждый цвет обозначен своим номером, всего цветов не более *109*.

В один из прекрасных дней в питомнике случилось чудо: все зверюшки выстроились в ряд в порядке возрастания цветов. Пользуясь случаем, лаборанты решили посчитать, сколько зверюшек разных цветов живет в питомнике, и, по закону жанра, попросили вас написать программу, которая поможет им в решении этой нелегкой задачи.

Для решения этой задачи используйте *std::vector* и библиотеку *algorithm*.

## Формат ввода

В первой строке входного файла содержится единственное число *N* *(0 ≤ N ≤ 105)* — количество зверюшек в Институте.

В следующей строке находятся *N* упорядоченных по неубыванию неотрицательных целых чисел, не превосходящих *109* и разделенных пробелами — их цвета.

В третьей строке файла записано число *M* *(1 ≤ M ≤ 100000)* — количество запросов вашей программе, в следующей строке через пробел записаны *M* целых неотрицательных чисел (не превышающих *109+1*).

## Формат вывода

Выходной файл должен содержать *M* строчек. Для каждого запроса выведите число зверюшек заданного цвета в питомнике.

## Пример

| **Ввод**  Скопировать ввод | **Вывод**  Скопировать вывод |
| --- | --- |
| 10  1 1 3 3 5 7 9 18 18 57  5  57 3 9 1 179 | 1  2  1  2  0 |

# C\*. Встреча

|  |  |
| --- | --- |
| Ограничение времени | 0.5 секунд |
| Ограничение памяти | 8Mb |
| Ввод | стандартный ввод или input.txt |
| Вывод | стандартный вывод или output.txt |

Напишите программу, которая определяет, сколько раз встречается заданное число *x* в данном массиве. Используйте последовательный контейнер и библиотеку алгоритмов для решения задачи. Старайтесь избегать циклов.

## Формат ввода

В первой строке задается одно натуральное число *N*, не превосходящее 1,000,000 — размер массива.

Во второй строке вводятся *N* чисел — элементы массива (целые числа, не превосходящие по модулю 1,000).

В третьей строке содержится одно целое число *x* , не превосходящее по модулю 1,000,000.

## Формат вывода

Вывести одно число — сколько раз встречается *x* в данном массиве.

## Пример

| **Ввод**  Скопировать ввод | **Вывод**  Скопировать вывод |
| --- | --- |
| 5  1 2 3 4 5  3 | 1 |

D\*. Список смежности

|  |  |
| --- | --- |
| Ограничение времени | 0.5 секунд |
| Ограничение памяти | 8Mb |
| Ввод | стандартный ввод или input.txt |
| Вывод | стандартный вывод или output.txt |

В этой задаче необходимо организовать неориентированный граф, на котором поддерживаются следующие операции:

1. *AddEdge(u, v)* — добавить в граф ребро между вершинами *(u, v)*.
2. *Vertex(u)* — вывести список вершин, смежных с вершиной *u*.

Петель и кратных ребер в графе нет. Для решения данной задачи используйте последовательные контейнеры из *STL*.

Формат ввода

В первой строке входного файла содержится целое число *N* *(1 ≤ N ≤ 106)* — количество вершин в графе.

В следующей строке находится целое число*K* *(0 ≤ K ≤ 106)* — число операций, затем идет описание операций — каждое в своей строке.

Операции имеют следующий формат: «1 *u* *v* » или «2 *u* », обозначающие соответственно операции *AddEdge(u, v)* и *Vertex(u)*.

Гарантируется, что суммарное количество чисел, которое будет необходимо вывести при выполнении всех операций Vertex не превосходит *2⋅ 105*.

Формат вывода

В выходной файл для каждой команды Vertex необходимо на отдельной строке вывести список смежных вершин указанной вершины. Вершины списка смежности нужно выводить в порядке добавления соответствующих ребер в граф.

Пример

| **Ввод**  Скопировать ввод | **Вывод**  Скопировать вывод |
| --- | --- |
| 4  4  1 1 2  1 2 3  2 2  1 1 3 | 1 3 |

Примечания

Подумайте, как реализовать команды выше в случае ориентированного графа или если в нем есть петли.

Подумайте, как реализовать хранение графа с кратными ребрами.

# E\*. Словарь

|  |  |
| --- | --- |
| Ограничение времени | 0.5 секунд |
| Ограничение памяти | 256Mb |
| Ввод | стандартный ввод или input.txt |
| Вывод | стандартный вывод или output.txt |

Вам дан словарь, состоящий из пар слов. Каждое слово является синонимом к парному ему слову. Все слова в словаре различны. Для каждого данного слова определите его синоним.

Для решения данной задачи используйте *std::unordered\_map*

## Формат ввода

Программа получает на вход количество пар синонимов *N* *(0 ≤ N ≤ 105)*. Далее следует *N* строк, каждая строка содержит ровно два слова-синонима.

Затем идет число *Q* *(1 ≤ Q ≤ 105)* — количество запросов к словарю. Далее на каждой следующей из *Q* строк идет слово, к которому надо вывести синоним.

## Формат вывода

Программа должна вывести синонимы к данным слову на отдельных строках.

## Пример

| **Ввод**  Скопировать ввод | **Вывод**  Скопировать вывод |
| --- | --- |
| 3  car plane  mouse cat  base stream  3  plane  stream  base | car  base  stream |

## Примечания

Подумайте, почему бы не использовать *std::map*. Используйте быстрый ввод.

# [https://contest.yandex.ru/contest/36912/download/E\*/](https://contest.yandex.ru/contest/36912/download/E*/)

# F\*. Банк

|  |  |
| --- | --- |
| Ограничение времени | 0.5 секунд |
| Ограничение памяти | 8Mb |
| Ввод | стандартный ввод или input.txt |
| Вывод | стандартный вывод или output.txt |

У банка есть клиенты. Каждый клиент имеет ровно один счет.

Напишите программу, которая будет выполнять последовательность запросов таких двух видов:

1. Начинается с числа 1, затем через пробел следует имя клиента (слово из латинских букв), далее через пробел идет сумма денег, которую клиент кладет или берет из счета в банке (целое число, не превышает по модулю 10000).
2. Начинается с числа 2, через пробел следует имя клиента. На каждый такой запрос программа должна ответить какая сумма в данный момент есть на счету заданного клиента. Если такое имя клиента пока ни разу не упоминалось в запросах вида 1, выводите вместо числа слово «ERROR».

В начале работы программы у всех клиентов на счету 0. Затем суммы могут становиться как положительными, так и отрицательными.

Обратите внимание, что в ситуации, когда клиент снял суммарно ровно столько же денег, сколько положил, сумма на счете становится равной 0, но, раз его имя уже встречалось, нулевое значение не является основанием выводить «ERROR».

Для решения задачи используйте ассоциативный контейнер.

Формат ввода

Первая строка стандартного входного потока количество запросов *N* *(1 ≤ N ≤ 105)*. Далее следуют *N* строк в каждой из которых описан один из двух описанных выше видов запроса.

Формат вывода

На каждый запрос 2-го вида нужно вывести текущее значение на счету заданного клиента (или слово «ERROR»).

Пример

| **Ввод**  Скопировать ввод | **Вывод**  Скопировать вывод |
| --- | --- |
| 7  1 asdf 3  1 zxcv 5  2 asdf  1 asdf -2  2 asdf  2 lalala  2 zxcv | 3  1  ERROR  5 |

# G\*. Частотность

|  |  |
| --- | --- |
| Time limit | 0.5 секунд |
| Memory limit | 32Mb |
| Input | stdin or input.txt |
| Output | stdout or output.txt |

## Legend

Дан текст. Выведите все слова, встречающиеся в тексте, по одному на каждую строку. Слова должны быть отсортированы по убыванию их количества появления в тексте, а при одинаковой частоте появления в лексикографическом порядке.

## Input format

Вводится текст — последовательность строк через пробел или перенос строки.

## Output format

Выведите ответ на задачу.

## Sample

| **Input**  Скопировать ввод | **Output**  Скопировать вывод |
| --- | --- |
| hi  hi  what is your name  my name is bond  james bond  my name is damme  van damme  claude van damme  jean claude van damme | damme  is  name  van  bond  claude  hi  my  james  jean  what  your |

## Notes

Указание. После того, как вы создадите словарь всех слов, вам захочется отсортировать его по частоте встречаемости слова. Желаемого можно добиться, если создать список, элементами которого будут пары из двух элементов: частота встречаемости слова и само слово. Например, *[(2, hi), (1, what), (3, is)]*. Тогда стандартная сортировка будет сортировать список пар, при этом кортежи сравниваются по первому элементу, а если они равны, то по второму. Это почти то, что требуется в задаче.

Набрать здесьОтправить файл

# H\*. База данных

|  |  |
| --- | --- |
| Ограничение времени | 0.5 секунд |
| Ограничение памяти | 32Mb |
| Ввод | стандартный ввод или input.txt |
| Вывод | стандартный вывод или output.txt |

Дана база данных о продажах некоторого интернет-магазина. Каждая строка входного файла представляет собой запись вида *покупатель-товар-количество* , где *покупатель* — имя покупателя (строка без пробелов), *товар* — название товараc (строка без пробелов), количество — количество приобретённых единиц товара.

Создайте список всех покупателей и для каждого покупателя подсчитайте количество приобретенных им единиц каждого вида товаров.

## Формат ввода

Во входном файле записано не более *105* строк в указанном формате.

Имена покупателей и названия товаров представляют собой строки из заглавных и строчных латинских букв не длиннее 10 символов. Количество товара в каждой покупке — натуральное число, не превышающее *106*.

## Формат вывода

Выведите список всех покупателей в лексикографическом порядке, после имени каждого покупателя выведите двоеточие, затем выведите список названий всех приобретенных данным покупателем товаров в лексикографическом порядке, после названия каждого товара выведите количество единиц товара, приобретенных данным покупателем.

Информация о каждом товаре выводится в отдельной строке.

## Пример

| **Ввод**  Скопировать ввод | **Вывод**  Скопировать вывод |
| --- | --- |
| Ivanov paper 10  Petrov pen 5  Ivanov marker 3  Ivanov paper 7  Petrov envelope 20  Ivanov envelope 5 | Ivanov:  envelope 5  marker 3  paper 17  Petrov:  envelope 20  pen 5 |

Набрать здесьОтправить файл

# I\*. Недешевый калькулятор

|  |  |
| --- | --- |
| Ограничение времени | 0.1 секунда |
| Ограничение памяти | 8Mb |
| Ввод | стандартный ввод или input.txt |
| Вывод | стандартный вывод или output.txt |

Студенты ФПМИ придумали новую версию калькулятора. Этот калькулятор берет с пользователя валюту за совершаемые операции. Стоимость каждой операции в *ФПМИ-коинах* равна 5% от числа, которое является результатом операции.

На этом калькуляторе требуется вычислить сумму *N* натуральных чисел (числа, так уж и быть, известны). Нетрудно заметить, что от того, в каком порядке мы будем складывать эти числа, иногда зависит, в какую *ФПМИ-коинов* нам обойдется вычисление суммы чисел (то есть оказывается нарушено великое правило «от перестановки мест слагаемых сумма не меняется»).

Например, пусть нам нужно сложить числа *10, 11, 12, 13*. Подумайте, как от порядка вычислений зависит ответ. Напишите программу, используя *std::priority\_queue* или функции для работы с пирамидой из *<algorithm>*, которая будет определять, за какую минимальную сумму *ФПМИ-коинов* можно найти сумму данных *N* чисел.

## Формат ввода

Во входном файле записано число *N* *(2 ≤ N ≤ 105)*. Далее идет *N* натуральных чисел, которые нужно сложить, каждое из них не превышает *104*.

## Формат вывода

В выходной файл выведите, сколько *ФПМИ-коинов* нам потребуется на нахождение суммы этих *N* чисел.

## Пример

| **Ввод**  Скопировать ввод | **Вывод**  Скопировать вывод |
| --- | --- |
| 4  10 11 12 13 | 4.6 |

## Примечания

Подумайте, какой еще контейнер *STL* мог бы решить эту задачу, почему рекомендована именно приоритетная очередь.

# J. Радио

|  |  |
| --- | --- |
| Ограничение времени | 1 секунда |
| Ограничение памяти | 32Mb |
| Ввод | стандартный ввод или input.txt |
| Вывод | стандартный вывод или output.txt |

### Пример 1

| **Ввод**  Скопировать ввод | **Вывод**  Скопировать вывод |
| --- | --- |
| GET  VOTE 1.1.1.1 1 -1 1  VOTE 1.1.1.1 2 1 2  VOTE 1.1.1.1 1 2 4  VOTE 1.1.1.1 1 4 20045  GET  GET  GET  VOTE 194.85.81.128 3 -1 20049  EXIT | 1 0  -2  0  -2  2  1 2  2 0  3 0  -2  OK |

### Пример 2

| **Ввод**  Скопировать ввод | **Вывод**  Скопировать вывод |
| --- | --- |
| GET  GET  GET  VOTE 192.168.0.1 2 2 11111111  EXIT | 1 0  2 0  3 0  1  OK |

https://contest.yandex.ru/contest/36912/download/J/