Задача A. Шашки

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

Сема — любитель игр на шахматной доске. Больше всего он любит играть в шашки. Сема решил написать программу, которая будет играть в шашки. По его задумке, она будет показывать, какие шашки можно взять на данном ходу. Ваша задача — реализовать эту функцию.

Напомним, что взять можно только по диагонали одним из четырех способов:

После того, как белая шашка перемещается на пустое поле, черная шашка снимается с доски и считается взятой. При этом, если после перемещения белой шашки, у нее вновь появляется возможность взять, то она продолжает свой ход. Аналогичны правила и для черных шашек.

Отметим что в рассматриваемом варианте игры в шашки отсутствует понятие «дамка», то есть возможности шашки по взятию не зависят от того, доходила она до последней горизонтали или нет.

Формат входного файла

Во первых восьми строках входного файла записаны по восемь символов из множества {«.», «B», «W»}, которые обозначают пустое поле, черную шашку, белую шашку соответственно. Во входном файле не более 12 шашек каждого цвета. Все шашки расположены либо на черных, либо на белых полях.

Формат выходного файла

В выходной файл выдайте поля, на которых стоят шашки, которые можно взять, если ходят белые или черные. Используйте формат вывода аналогичный показанному в примерах.

Отсортируйте поля сначала по первой координате (измеряется по вертикали), а при равенстве первых — по второй (измеряется по горизонтали).

Учитывайте, что первый символ первой строки входного файла соответствует полю (1, 1), последний символ первой строки — полю (1, 8), первый символ последней строки — полю (8, 1), последний символ последней строки — (8, 8).

Пример

checkers.in	checkers.out
.W.W.W.W W.W.W.W. .W.W.W.W .... B.B.B.B. .B.B.B.B B.B.B.B.	White: 0 Black: 0
.W.W.W.W W.W.W.W. .....W.. W.W...W. .B.B.... B...B.B. .B.B.B.B B.B...B.	White: 3 (5, 2), (5, 4), (7, 4) Black: 1 (4, 3)

В первом примере никакая шашка не может брать. Во втором примере белая шашка, стоящая на $(4, 1)$ может взять сначала черную шашку на поле $(5, 2)$ и переместившись на поле $(6, 3)$ взять черную шашку, стоящую на поле $(5, 4)$. Также с поля $(6, 3)$ можно взять еще одну черную шашку, стоящую на поле $(7, 4)$, при этом придется двигаться другим путем. Черные же могут взять лишь белую шашку, которая стоит на поле $(4, 3)$.

Задача B. Пирожные

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

Санта Клаус планирует принести подарки на новый год n ребятам. У него есть m пирожных и он собирается подарить каждому ребенку несколько пирожных. Однако неожиданно возникла проблема. Детям не нравится, когда кому-то достается больше пирожных, чем ему.

Каждый ребенок характеризуется своей *жадностью*, жадность i-го ребенка равна g_i. Если a_i ребят получат больше пирожных чем i-й, то его неудовлетворенность будет равна $g_i a_i$.

Теперь у Санты проблема — надо поделить пирожные между ребятами так, чтобы суммарная неудовлетворенность была минимальная. Каждый ребенок должен получить хотя бы одно пирожное. Санта собирается раздать все m пирожных. Помогите ему распределить их между ребятами!

Формат входного файла

Первая строка входного файла содержит числа n и m ($1 \le n \le 30$, $n \le m \le 5000$). Вторая строка содержит n целых чисел $g_1, g_2, \ldots, g_n$ ($1 \le g_i \le 10^7$).

Формат выходного файла

На первой строке выходного файла выведите минимальную возможную суммарную неудовлетворенность. Вторая строка должна содержать n целых чисел — сколько пирожных следует дать каждому ребенку. Если решений несколько, выведите любое.

Пример

cookies.in	cookies.out
3 20 1 2 3	2 2 9 9
4 9 2 1 5 8	7 2 1 3 3

Задача C. ePig

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

Андрей и Аня разрабатывают новую P2P сеть для обмена файлами, они назвали свою сеть *ePig*. В этой задаче вам предлагается просимулировать работу сети при распространении одного большого файла.

Пусть сетью пользуются n клиентов, пронумерованных от 1 до n. Исходно файл целиком доступен на клиенте номер 1. Остальные клиенты хотели бы получить этот файл. Для оптимизации процесса файл разбивается на k идентичных фрагментов, пронумерованных от 1 до k. Передача файла состоит из нескольких раундов. Каждый раунд занимает одну минуту, за время раунда каждый клиент может получить ровно один фрагмент и/или передать ровно один фрагмент. После того как клиент скачивает фрагмент файла, он может раздавать его другим клиентам.

Перед каждым раундом каждый клиент решает, какой фрагмент он будет запрашивать. Клиент запрашивает фрагмент, который предоставляется наименьшим числом клиентов (разумеется, кроме тех фрагментов, которые у него уже есть). Если таких фрагментов несколько, он выбирает тот, у которого минимальный номер.

После этого клиенты делают запросы на фрагменты. Каждый клиент выбирает другого клиента, у которого он запрашивает выбранный фрагмент. Если несколько клиентов предоставляют необходимый фрагмент, то выбирается клиент, у которого минимальное количество предоставляемых им фрагментов. Если и таких клиентов несколько, выбирается клиент с минимальным номером.

Каждый клиент рассматривает все запросы, которые к нему поступили, и выбирает один их них. Клиент X удовлетворяет тот из запросов, который приходит от самого ценного клиента. Ценность клиента определяется количеством фрагментов, которые клиент X получал от него ранее. Если имеется несколько клиентов с одинаковой ценностью, то фрагмент отдается тому из клиентов, у которого перед раундом имеется минимальное количество фрагментов. Если и таких клиентов несколько, то фрагмент отдается клиенту с минимальным номером.

После того, как выбрано, какие запросы будут удовлетворены, начинается раунд. Клиенты, запросы которых были отклонены, ничего не скачивают в этот раунд, а остальные клиенты скачивают запрошенные фрагменты. После этого начинается новый раунд, и т. д.

По заданным n и k для каждого клиента определите число раундов, которое потребуется, чтобы он получил файл целиком.

Формат входного файла

Первая строка входного файла содержит два целы числа: n и k ($2 \le n \le 100$, $1 \le k \le 200$).

Формат выходного файла

Выведите $n-1$ число — для каждого клиента кроме первого выведите одно число — количество раундов перед тем как он скачает файл целиком.

Пример

epig.in	epig.out
3 2	3 3

Распространение файла в данном случае происходит следующим образом: в первом раунде клиенты 2 и 3 запрашивают фрагмент 1 от клиента 1. Удовлетворяется запрос от клиента 2. После этого клиенты 2 и 3 запрашивают фрагмент 2 у клиента 1. Удовлетворяется клиент 3. Наконец на третьем раунде клиент 2 запрашивает фрагмент 2 у клиента 3, а клиент 3 запрашивает фрагмент 1 у клиента 3, оба запроса удовлетворяются и у каждого теперь есть целый файл.

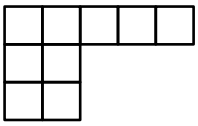
Задача D. Неприводимые диаграммы Юнга

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

Диаграммы Юнга используются для того, чтобы изобразить разбиение числа на слагаемые. Разбиение числа n на слагаемые представляет собой сумму вида $n = m_1 + m_2 + \ldots + m_k$, где $m_1 \geq m_2 \geq \ldots \geq m_k$.

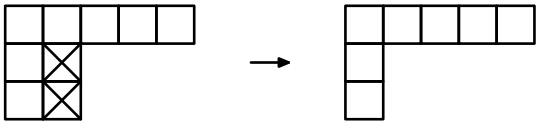
Диаграмма состоит из n квадратиков, организованных в виде k рядов, где k количество слагаемых в разбиении. Ряд, соответствующий числу m_i, содержит m_i квадратиков. Все ряды выровнены по левому краю и упорядочены от более длинного к более короткому.

Например, диаграмма Юнга, приведенная на рисунке, соответствует разбиению $9 = 5 + 2 + 2$.

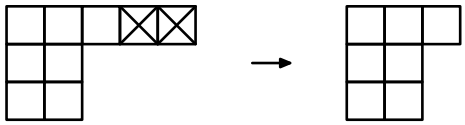


Рассмотрим один метод преобразования диаграмм Юнга. Разрешается выбрать любые два соседних квадратика и удалить их. При этом накладывается следующее ограничение: после преобразования оставшийся набор квадратиков должен быть корректной диаграммой Юнга, верхний левый квадратик которой расположен в том же месте, что и у исходной диаграммы (либо должна остаться пустая диаграмма).

Например, удалив последние квадратики второго и третьего ряда из приведенной выше диаграммы, мы получаем диаграмму для разбиения $7 = 5 + 1 + 1$.



Удаляя два последних квадратика первого ряда из этой исходной диаграммы, мы получаем диаграмму для разбиения $7 = 3 + 2 + 2$.



И еще один способ преобразовать эту диаграмму — удалить последний ряд целиком.

Диаграмма, которую нельзя преобразовать указанным способом, называется *неприводимой*. В частности, пустая диаграмма Юнга является неприводимой.

Каждую диаграмму можно преобразовывать до тех пор, пока она не станет неприводимой. Вообще говоря, может быть несколько способов преобразовать диаграмму к неприводимой. По заданной диаграмме Юнга найдите все неприводимые диаграммы, в которые ее можно преобразовать.

Формат входного файла

Первая строка входного файла содержит число k — количество рядов в диаграмме ($1 \leq k \leq 100\,000$). Вторая строка содержит k целых чисел: $m_1, m_2, \ldots, m_k$. Сумма $n = m_1 + m_2 + \ldots + m_k$ не превышает 10^8.

Формат выходного файла

На первой строке выходного файла выведите число l — количество неприводимых диаграмм Юнга, к которым можно преобразовать заданную. Следующие l строк должны описывать эти диаграммы. Каждая строка должна начинаться с числа t — количества рядов в соответствующей диаграмме. Далее должно следовать t чисел — количество квадратиков в соответствующих рядах.

Пример

irreducible.in	irreducible.out
3 5 2 2	1 1 1
1 2	1 0

Задача E. Объявление массивов

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

В языке *Java* для удобства работы любой объявленный массив является объектом, который содержит некоторую вспомогательную информацию и собственно сам массив.

Рассмотрим внутреннюю структуру такого объекта. В нем содержится 16 байт вспомогательной информации, 4 байта уходит на хранение длины массива, а затем сам массив, который занимает $length \cdot size$ памяти, где $length$ — это его длина, а $size$ — размер объектов хранящихся в массиве. Если мы объявляем в программе двухмерный массив, то у нас создаться объект, который будет содержать в себе массив, каждый элемент которого будет являться одномерным массивом. Так же не следует забывать, что на каждый объект создается внешняя ссылка размером 4 байта.

Аналогично, при объявлении массивов размерности k будет создан массив, каждым элементом которого является массив размерности $k-1$.

Ваша задача — по данным о размерах примитивных типов, из которых состоят массивы, выяснить, сколько памяти занимает каждый из заданных массивов.

Формат входного файла

В первой строке содержится количество различных примитивных типов N ($1 \le N \le 100$). В следующих N строках содержаться названия примитивных типов, состоящие из маленьких латинских букв, длиной не более 30 символов, и через пробел размер типа в байтах. Размеры типов не превосходят $2^{31}-1$ байт.

Следующая строка сдержит число объявленных массивов M ($1 \le M \le 1000$). В следующих M строках содержаться описания массивов в формате «$type\ name[number_1]\dots[number_k]$;», где $type$ — один из примитивных типов описаных выше, $name$ — имя массива, состоящее из маленьких латинских букв, длиной не более 30 символов, $1 \le number_i \le 1000$. Размерность массива не превышает 50.

Формат выходного файла

В M строках выходного файла объем памяти в байтах, занимаемой соответствующим массивом.

Пример

java.in	java.out
4 byte 1 short 2 int 4 long 8 4 byte a[1]; short b[2][2][2]; int c[100][100]; long a[177];	25 184 42424 1440

Первый массив занимает 4 байта на внешнюю ссылку, 16 вспомогательных байт, 4 байта на хранение длины массива и 1 байт на сам массив.

Третий массив занимает 4 байта на внешнюю ссылку, 16 вспомогательных байт, 4 байта на хранение длины массива и $100 \cdot size$ байт на массив, где $size$ — размер одномерного массива из 100 чисел типа int. В данном случае $size = 4 + 16 + 4 + 4 * 100$. Итого 42424 байт.

Задача F. Симпатичные последовательности

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

Рассмотрим последовательность $a_1, a_2, \ldots, a_n$ неотрицательных целых чисел. Обозначим как $c_{i,j}$ количество появлений числа i среди чисел $a_1, a_2, \ldots, a_j$. Будем называть последовательность *k-симпатичной*, если для всех $i_1 < i_2$ и для всех j выполнено условие: $c_{i_1,j} \geq c_{i_2,j} - k$.

По заданной последовательности $a_1, a_2, \ldots, a_n$ и числу k, найдите ее максимальный префикс, который является k-симпатичным.

Формат входного файла

Первая строка входного файла содержит числа n и k ($1 \leq n \leq 200\,000$, $0 \leq k \leq 200\,000$). Вторая строка содержит n целых чисел в диапазоне от 0 до n.

Формат выходного файла

Выведите максимальное l такое что последовательность $a_1, a_2, \ldots, a_l$ является k-симпатичной.

Пример

nice.in	nice.out
10 1 0 1 1 0 2 2 1 2 2 3	8
2 0 1 0	0

Задача G. Обмен пакетами

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

Вася получает доступ в Интернет с помощью мобильного телефона. Однако такая связь очень нестабильна, поэтому каждый раз после подключения ему приходиться проверять подключение. Для этого он просто в командной строке набирает примерно следующую фразу:

«`ping` *name*»,

где *name* — это имя удаленного сервера, который точно находиться в сети. Затем идет обмен пакетами с сервером и выдается статистика. Однако у Васи недавно сломался модуль, отвечающий за подсчет и вывод статистики. Вам надо будет помочь Васе — написать аналогичный модуль.

После вызова команды «`ping`» на удаленный сервер по очереди посылаются 4 пакета по 32 байта. Как только удаленный сервер получил пакет, он отвечает на него. Если пакет не уложился в определенное время (он должен дойти до удаленного сервера и вернуться обратно) в силу тех или иных причин (низкая скорость, отсутствие подключения и т.д.), он считается утерянным.

Дана информация обо всех 4 пакетах. Требуется определить количество потерянных пакетов, максимальное, минимальное и среднее время обмена одного пакета.

Формат входного файла

Входной файл содержит ровно 5 строк. В первой строке находится фраза «`ping` *name*», где *name* — это имя сервера. Имя сервера представляет собой `IP` адрес. `IP`-адрес — это 4 однобайтных числа (т.е. числа от 0 до 255), отделенные друг от друга точкой.

В каждой из следующих 4 строк содержится либо фраза «`Time out`», если пакет считается утерянным, либо «Reply from *name* Time=*number*», где *name* — это имя удаленного сервера, а *number* — время за которое вернулся пакет (*number* — целое число, $0 \le number \le 10^4$).

Формат выходного файла

Выведете статистику по обмену пакетами с удаленным сервером. Следуйте формату приведенному в примере. Среднее время округлите до целого числа по математическим правилам. Если все 4 пакета утеряны, то выведите только первые две строки.

Пример

ping.in

`ping 209.85.135.147`
`Time out`
`Reply from 209.85.135.147 Time=100`
`Reply from 209.85.135.147 Time=300`
`Reply from 209.85.135.147 Time=200`

ping.out

`Ping statistics for 209.85.135.147:`
`  Packets: Sent = 4 Received = 3 Lost = 1 (25% loss)`
`Approximate round trip times:`
`  Minimum = 100 Maximum = 300 Average = 200`

Задача H. Длиннейшая общая подпара

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

Будем называть пару строк (α, β) *подпарой* строки γ если $\gamma = \gamma_1\alpha\gamma_2\beta\gamma_3$ для некоторых (возможно пустых) строк γ_1, γ_2 и γ_3. Длиной пары строк будет называть сумму длин составляющих ее строк: $|(\alpha, \beta)| = |\alpha| + |\beta|$.

По заданным двум строкам ξ и η найдите их длиннейшую общую подпару, то есть такую пару строк (α, β), что она является подпарой как ξ, так и η, и ее длина максимальная.

Формат входного файла

Входной файл содержит две непустые строки ξ и η, состоящие из маленьких букв латинского алфавита. Длина каждой из строк не превышает 3000.

Формат выходного файла

Выведите α на первой строке выходного файла и β на второй строке.

Пример

subpair.in	subpair.out
abacabadabacaba acabacadacabaca	acaba abaca
ab bc	b

Задача I. Окопы

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

Однажды Андрей и Петя решили пострелять в друг друга из пистолетов. Нет, конечно, не из настоящих, а из игрушечных. Для того, чтобы игра стала еще более интересной, каждый из них вырыл себе окоп. Окоп представляет собой отрезок, соединяющий две точки. Известно, что окопы Андрея и Пети не пересекаются, более того, они вообще не имеют общих точек.

Теперь им предстоит выбрать «линию фронта», такую прямую, которую в процессе игры пересекать запрещается. Только вот проблема — окопы уже вырыты, а ребята не могут сообразить, как им провести линию, так чтобы окопы оказались по разные стороны от нее и не имели общих точек с ней. Так как рытье окопов дело трудоемкое, то они попросили помощи у Вас.

Формат входного файла

Входной файл содержится несколько тестов. Каждый тест описывается двумя строками.

Первая строка теста содержит 4 целых числа x_1, y_1, x_2, y_2 — координаты концов отрезка, задающего первый окопа. Вторая строка теста в аналогичном формате описывает второй окоп.

Входной файл заканчивается двумя строчками, каждая из которых содержит 4 нуля.

Тесты разделены пустой строкой.

Все координаты не превышают 10^4 по абсолютной величине.

Формат выходного файла

Для каждого теста в отдельной строке выведите числа a, b, c — коэффициенты уравнения прямой $ax + by + c = 0$, разделяющей окопы. Числа a, b, c должны быть целыми и не должны превышать 10^9 по абсолютной величине.

Примеры

trenches.in	trenches.out
0 1 1 2 0 -1 1 0 0 1 -1 2 0 -1 -1 0 0 0 0 0 0 0 0 0	1 -1 0 1 1 0