# 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<br>W.W.W.W.<br>.W.W.W.W<br>........<br>........<br>B.B.B.B.<br>.B.B.B.B<br>B.B.B.B. | White: 0<br>Black: 0 |
| .W.W.W.W<br>W.W.W.W.<br>.....W..<br>W.W...W.<br>.B.B....<br>B...B.B.<br>.B.B.B.B<br>B.B...B. | White: 3<br>(5, 2), (5, 4), (7, 4)<br>Black: 1<br>(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. Бег по эскалатору

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

Пусть $N$ человек бегут вниз по эскалатору, причем $i$-ый пробегает одну ступеньку за $t_i$ секунд. По технике безопасности бега по эскалатору, на эскалаторе запрещены «обгоны», то есть если человек $A$ в процессе бега догнал человека $B$, который бежит с более низкой скоростью, то далее, до конца эскалатора, человек $A$ бежит со скоростью человека $B$. Однако ступени эскалатора таковы, что на них может помещаться несколько человек одновременно.

Ваша задача написать программу, которая поможет работникам станции рассчитать, когда закончит свой бег по эскалатору каждый бегущий человек.

## Формат входного файла

В первой строке входного файла записано число $N$ ($1 \le N \le 10^5$). В следующих $N$ строках перечислены пары чисел $t_i$, $w_i$ ($1 \le t_i, w_i \le 10^6$) — время пробега одной ступени и количество ступеней до конца эскалатора. Гарантируется, что изначально всем людям осталось бежать различное число ступеней.

## Формат выходного файла

В $i$-той строке выходного файла запишите время в секундах, через которое $i$-ый человек сойдет с эскалатора.

## Примеры

| escalator.in | escalator.out |
|---|---|
| 3<br>2 10<br>3 11<br>1 12 | 20<br>33<br>33 |

# Задача C. Финал ACM ICPC

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

Ежегодно в Санкт-Петербурге, Барнауле и некоторых городах ближнего зарубежья проходят соревнования по программированию. Эти соревнования проходят в рамках студенческого чемпионата мира по программированию, организованного одной из самых авторитетных ассоциаций ACM (Association for Computing Machinery). На этих соревнованиях проходит отбор команд с Северо-Восточного Европейского Региона NEERC (North-Eastern European Regional Contest).

Ежегодно перед организаторами соревнований встает проблема определения команд, которые будут приглашены к участию в финале чемпионата мира по программированию. По новым правилам в финал проходят не более $N$ команд, представляющих NEERC. Кроме этого, от одного вуза не может проходить более чем $k$ команд. При этом из всех таких множеств выбирается то, в котором сумма мест занятых этими командами в полуфинальных соревнованиях минимальная возможная.

Ваша задача по итоговому протоколу полуфинальных соревнований и числам $N$ и $k$ определить, какие команды будут приглашены к участию в финале чемпионата мира.

## Формат входного файла

В первой сроке входного файла находится три натуральных числа $P$ $(1 \le P \le 100000)$ — количество команд, принявших участие в полуфинале, $N$ $(1 \le N \le P)$ и $k$ $(1 \le k \le P)$.

В следующих $P$ строках, по одному в строке перечислены названия университетов, команды которых заняли соответствующие места. Название университета содержит строчные и прописные латинские буквы и пробелы. Длина названия университета не превышает 30 символов. В следующей строке перечислены номера команд соответствующих университетов. Таким образом если название университета записано в $i$ - той строке $(2 \le i \le P+1)$, то эта команда заняла $i-1$ место на полуфинале и имеет номер, записанный на $i-1$ месте в $P+2$ строке.

## Формат выходного файла

В выходной файл выведите названия команд, приглашенных к участию в финале чемпионата мира по программированию, упорядоченных по месту, занятому на полуфинале. В качестве названия команды выведите название университета и через пробел `#номер команды`.

## Пример

| final.in | final.out |
|---|---|
| `9 5 2`<br>`Fantasy University`<br>`Crazy University`<br>`Fantasy University`<br>`Fantasy University`<br>`Very Good U`<br>`Good U`<br>`Very Good U`<br>`Crazy University`<br>`Good U`<br>`1 1 2 3 2 1 1 2 2` | `Fantasy University #1`<br>`Crazy University #1`<br>`Fantasy University #2`<br>`Very Good U #2`<br>`Good U #1` |

# Задача D. 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 «$type\ name[number_1] \ldots [number_k];$», 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<br>byte 1<br>short 2<br>int 4<br>long 8<br>4<br>byte a[1];<br>short b[2][2][2];<br>int c[100][100];<br>long a[177]; | 25<br>184<br>42424<br>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 E. Прыгучее путешествие

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

Кузнечик Тема очень любит прыгать по полянке, на которой он живет. Но однажды беда обрушилась на его голову. На его полянке поселилась злобная лягушка, которая была бы не против полакомиться кузнечиком Темой. Однако Тема решил не унывать и все-таки продолжать прыгать и резвиться на полянке. Однако теперь ему надо делать это весьма осторожно. Для этого он хочет составить план своего путешествия.

Тема хочет посетить точки $A_1 = (x_1, y_1), A_2 = (x_2, y_2), \ldots, A_n = (x_n, y_n)$ (можно считать, что изначально он находиться в точке $A_1$, затем прыгает в точку $A_2$, потом в $A_3$ и т.д.). Так же ему известно, что лягушка живет в точке $B = (x, y)$ и имеет длину языка $l$. Таким образом если после приземления Тема оказывается на расстоянии не более $l$ от точки $B$, то лягушка его съест, то есть лягушка может съесть Тему только, когда он находится на земле. Во время полета съесть Тему лягушка не может.

Ваша задача помочь Теме определить, будет ли он съеден лягушкой или нет.

## Формат входного файла

В первой строке сходного файла записано число $N$ ($1 \le N \le 100000$), координаты точки $B$ и число $l$ ($0 \le l \le 5000$). В следующих $N$ строках записаны координаты точек $A_i$. Все координаты — это целые числа по модулю не превосходящие $10^4$.

## Формат выходного файла

Если кузнечик успешно завершит свое путешествие в точке $A_n$ и не будет съеден в ней выведите строку «Yes», иначе выведите номер точки, после приземления в которую Тему съест лягушка.

Если Тема может быть съеден лягушкой в нескольких точках, выведите точку, заданную раньше во входном файле (лягушка, разумеется, использует свой первый шанс, чтобы съесть кузнечика).

## Примеры

| **jump.in** | **jump.out** |
|---|---|
| 3 0 0 1<br>2 0<br>1 1<br>0 1 | 3 |
| 2 0 0 100<br>1 1<br>2 2 | 1 |
| 3 0 0 1<br>1 1<br>2 2<br>3 3 | Yes |

# Задача F. 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 G. Последовательность чисел

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

Рассмотрим рекуррентно заданную последовательность чисел $S_i$:
1. $S_1 = 1$
2. $S_i = S_{i-1}, i, S_{i-1}$ для $i \geq 2$.

Ваша задача написать программу, которая будет находить число стоящее на $k$-ой позиции в последовательности $S_p$.

## Формат входного файла

В первой строке входного файла записано число тестов $T$, $(1 \leq T \leq 1000)$. В следующих $T$ строках записаны пары чисел $k_i$, $p_i$, $(1 \leq k_i, p_i \leq 2^{63} - 1,\ 1 \leq i \leq T)$.

## Формат выходного файла

Если в $S_{p_i}$ существует $k_i$-тое число, то вывести его, в противном случае вывести «No solution», без кавычек.

## Примеры

| **sequence.in** | **sequence.out** |
|---|---|
| 4<br>1 1<br>2 1<br>1 2<br>2 2 | 1<br>No solution<br>1<br>2 |

# Задача H. 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<br>0 -1 1 0<br><br>0 1 -1 2<br>0 -1 -1 0<br><br>0 0 0 0<br>0 0 0 0 | 1 -1 0<br>1 1 0 |