
"First, solve the problem. Then, write the code" John Johnson.

# Programa-Me 2016
## Final Nacional
## Problemas

Patrocinado por





Ejercicios realizados por


Universidad Complutense
de Madrid

Realizado en la **Facultad de Informática (U.C.M.)**
3 de junio de 2016

3 de junio de 2016
http://www.programa-me.com

# Listado de problemas



Autores de los problemas:

- Marco Antonio Gómez Martín (Universidad Complutense de Madrid)
- Pedro Pablo Gómez Martín (Universidad Complutense de Madrid)
- Alberto Verdejo López (Universidad Complutense de Madrid)

# A
# Reina del súper

Hace tiempo, Ismael se enamoró perdidamente de una cajera del supermercado. No se cansa de espiarla entre la sección de ofertas y menaje del hogar.

Tiene la tarjeta echando humo, porque aunque tenga la nevera llena, todas las tardes baja a comprar cualquier cosa con tal de volver a verla. Y da igual cuánta gente haya; él siempre se pone a esperar en la fila de su caja.

Pero al llegar hoy al súper se ha llevado una desagradable sorpresa. Al cerebrito de turno que gestiona el supermercado le ha dado por instaurar la "fila única". Ahora en lugar de haber filas independientes para cada caja, hay una única fila para todas y cuando una caja se queda vacía, el primero que ocupa la fila va a ella.

El gerente del súper seguramente piense que ahora los clientes quedarán más satisfechos con el servicio proporcionado porque saben que nunca esperarán de más, pero a Ismael le han destrozado la vida. Ya no tiene la garantía de que le atienda su reina del súper particular cuando le toca pagar.

Sólo le queda una alternativa. Cuando se aproxima a la fila única, puede estimar cuánto tardará cada cliente en pasar por la caja en base a la cantidad de productos en el carrito y averiguar entonces en qué caja le tocará para ver si es la de su amada o no.

## Entrada

La entrada estará compuesta por distintos casos de prueba, cada uno de ellos representando el estado del supermercado uno de los días en los que Ismael va a comprar.

Cada caso de prueba consta de dos líneas. La primera contiene el número $n$ de cajas abiertas en ese momento ($1 \leq n \leq 5$) y el número $c$ de clientes esperando ($1 \leq c \leq 1.000$). A continuación viene una línea con $c$ números positivos que indican el número de segundos que tardará cada cliente en ser atendido. El primer número se corresponde con el tiempo de la primera persona de la fila única.

Al último caso de prueba le sigue uno con 0 cajas y 0 clientes que no debe procesarse.

## Salida

Para cada caso de prueba se escribirá una línea con el número de la caja en la que será atendido Ismael si se pone a esperar en ese momento.

Ten en cuenta que las cajas están numeradas de la 1 a la $n$ y que en caso de quedar dos cajas libres a la vez, el primer cliente irá a la caja con menor número.

## Entrada de ejemplo

```
2 2
10 5
2 2
5 10
3 2
5 10
0 0
```

## Salida de ejemplo

```
2
1
3
```

# B
# ¡No lo puedes saber!

— He pensado un número entre 1 y 10 (incluídos), ¿intentas adivinarlo? Yo te contestaré simplemente "menor" o "mayor o igual" hasta que seas capaz de saber cuál es.

— ¡Acepto el reto! ¿Es el 1?

— Mayor o igual.

— ¿Es el 2?

— Menor

— ¡Lo tengo!

— ¡Vaya! ¡Qué rapidez! Venga, otra vez. Ahora un número entre 1 y 1000.

— ¿El 500?

— Menor

— ¿El 400?

— Mayor o igual.

— ¡Lo tengo!

— ¿Cómo que lo tienes? ¡No puede ser!

## Entrada

La entrada estará compuesta por distintas partidas al juego anterior. Cada una de esas partidas ocupará dos líneas. La primera tendrá tres números, *ini*, *fin* y $n$ con $1 \leq ini \leq n \leq fin \leq 250.000$ indicando el rango de valores entre los que se puede elegir el número y el valor oculto elegido.

La segunda línea comienza con el número $k$ de preguntas que hace el segundo jugador, al que le siguen $k$ números con las hipótesis. Se garantiza que éstas están siempre en el intervalo $[ini, fin]$.

El último caso de prueba va seguido de una línea con tres ceros que no debe procesarse.

## Salida

Para cada caso de prueba se escribirá `LO SABE` si al final de todas las preguntas el jugador puede saber el número oculto y `NO LO SABE` si es imposible que lo sepa con certeza.

Se asume que la persona que contesta a las preguntas nunca miente.

## Entrada de ejemplo

```
1 10 1
2 1 2
1 1000 450
2 400 500
0 0 0
```

## Salida de ejemplo

```
LO SABE
NO LO SABE
```

# C
# Siete de un golpe

En la adaptación libre que hizo Walt Disney del cuento *El sastrecillo valiente* de los hermanos Grimm un sastre está siendo molestado por siete moscas. La fortuna hace que consiga eliminar las siete de un solo golpe de matamoscas[1].

La dificultad de matar de un solo golpe a siete moscas apoyadas en una mesa depende fundamentalmente de tres cosas: el número de moscas que haya en la mesa, el tamaño del matamoscas y lo rápido que éstas sean para levantar el vuelo.

Podemos imaginarnos la mesa dividida en pequeños cuadrados todos del mismo tamaño y que pueden o no tener una mosca dentro. Y podemos tener un matamoscas de tamaño $N \times M$ que cuando se coloca sobre la mesa, liquida a todas las moscas que hay en esos $N \times M$ cuadrados.

Para ver cómo de extraordinaria es la hazaña del sastre, queremos averiguar en cuántas posiciones podemos colocar el matamoscas sobre la mesa (orientado siempre en horizontal) para matar distintas moscas de un golpe.

## Entrada

La entrada estará compuesta por distintos casos de prueba. Cada uno de ellos comienza con una línea con cuatro números: el número de cuadrados en horizontal ($tx$) y vertical ($ty$) en los que está dividida la mesa (como mucho 1.000) y el tamaño del matamoscas, o número de cuadrados que cubre en horizontal ($1 \leq mx \leq tx$) y vertical ($1 \leq my \leq ty$). A continuación vienen $ty$ líneas cada una con $tx$ caracteres que indican si en esa celda hay mosca (`X`) o no (`.`).

La entrada termina con un tablero y matamoscas de tamaños $0 \times 0$, que no debe procesarse.

## Salida

Por cada caso de prueba se escribirá una línea con ocho números enteros; el primero de ellos contiene el número de posiciones en donde el matamoscas no mataría ninguna mosca; el siguiente indica las posiciones en las que moriría una única mosca, después dos moscas, y así sucesivamente hasta terminar con el último número que marca las posiciones en las que el sastre mataría a siete moscas de un golpe.

Ten en cuenta que el matamoscas siempre se coloca con la misma orientación y que debe entrar todo entero en la mesa. Además, podría haber posiciones en las que el número de moscas muertas sea superior a siete.

## Entrada de ejemplo

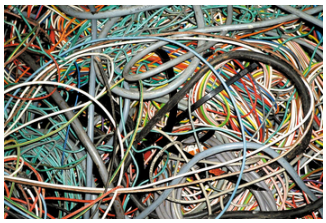
```
3 3 2 2
...
.X.
...
4 3 4 2
X.XX
X.XX
..XX
4 3 3 3
XXX.
XXX.
XXX.
0 0 0 0
```

[1] En el cuento original el número de moscas eran "legiones" y, dice, aniquila por lo menos a veinte utilizando un paño. Vemos, pues, que la película de Walt Disney dista mucho del original.

## Salida de ejemplo

```
0 4 0 0 0 0 0 0
0 0 0 0 0 1 1 0
0 0 0 0 0 0 1 0
```

# D
# Conectando cables



El caos más absoluto se ha apoderado de la caja en la que se guardan los cables USB en el taller de *hardware*. La situación es ya insostenible, así que toca ordenarlos un poco.

Como la tarea es pesada, se te ha ocurrido un juego: intentar conectar todos los cables unos con otros formando un gran círculo... aunque ni siquiera tienes claro que eso pueda hacerse, porque depende de cuántos cables de cada tipo haya en la caja...

## Entrada

La entrada comienza con una línea con el número de casos de prueba que vendrán a continuación.

Cada uno de los casos representa un escenario en el que hay que intentar conectar todos los cables. Contiene dos líneas, una con el número de cables que hay en la caja (al menos uno y no más de 10.000) y otra con una descripción de cada cable. Esa descripción no es más que dos letras, una por cada extremo, donde `M` significa conector USB macho y `H` conector hembra.

## Salida

Para cada caso de prueba hay que escribir `POSIBLE` si pueden conectarse todos los cables para formar un círculo o `IMPOSIBLE` en caso contrario.

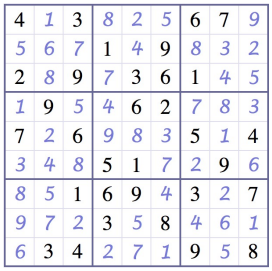
## Entrada de ejemplo

```
4
1
HM
1
HH
2
HM MH
3
HM HH MM
```

## Salida de ejemplo

```
POSIBLE
IMPOSIBLE
POSIBLE
POSIBLE
```

# E
# Sudokus correctos




El sudoku es un pasatiempo lógico que consiste en rellenar una cuadrícula de 9×9 casillas dividida en nueve regiones 3×3 (las separadas con líneas más gruesas en la imagen) con los números del 1 al 9 de tal forma que no se repitan números en ninguna fila, columna o región. El sudoku inicialmente se presenta con algunas casillas ya rellenas, a modo de *pistas*, y el jugador debe deducir los valores de las casillas vacías. Si el sudoku está bien planteado, la solución es única.

Dado un sudoku completamente relleno, ¿sabrías construir un programa que comprobara si es correcto (es decir, cada fila, columna o región contiene los números del 1 al 9 exactamente una vez)?

## Entrada

La entrada comienza con un número que representa el número de casos de prueba que vendrán a continuación.

Cada caso de prueba está formado por 9 líneas, cada una con 9 números entre el 1 y el 9 separados por espacios, que representan un sudoku completamente relleno.

## Salida

Para cada caso, se escribirá una línea con la palabra `SI` si el sudoku ha sido resuelto correctamente, y `NO` en caso contrario.

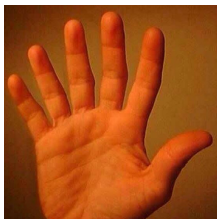
## Entrada de ejemplo

```
1
4 1 3 8 2 5 6 7 9
5 6 7 1 4 9 8 3 2
2 8 9 7 3 6 1 4 5
1 9 5 4 6 2 7 8 3
7 2 6 9 8 3 5 1 4
3 4 8 5 1 7 2 9 6
8 5 1 6 9 4 3 2 7
9 7 2 3 5 8 4 6 1
6 3 4 2 7 1 9 5 8
```

## Salida de ejemplo

```
SI
```

# F
# El hombre con seis dedos



El otro día me presentaron a un hombre y al darnos la mano noté algo extraño. ¡Luego me di cuenta de que tenía seis dedos! La cosa me sorprendió tanto que he estado investigando y resulta que no es tan extraño que una persona nazca con más de cinco dedos en una mano o un pie. La anomalía genética se denomina *polidactilia* y se estima que 1 de cada 500 bebés nace con ella. Lo que ocurre es que normalmente el dedo extra es muy pequeño y termina siendo extirpado sin más consecuencias. Lo que sí es más extraño es que el dedo se encuentre completo y sea funcional. Eso es lo que le ocurría a este hombre.

Continuando mi investigación he dado con una página web donde se han recopilado los años donde se conoce que nació una persona con algún dedo de más, completo y funcional. Y estoy interesado en construir un programa para averiguar en qué siglo nacieron más personas así. Ya puestos, sería mejor tener un programa más general, que sirviera también para averiguar la decada con más nacimientos, o el lustro, etc.

## Entrada

El programa recibirá por la entrada estándar una serie de casos de prueba. Cada caso consta de dos líneas. En la primera aparecen dos números: el número $N$ de nacimientos de los que tenemos constancia (entre 1 y 200.000) y el número $A$ de años del periodo en el que estamos interesados (un número mayor que cero). En la segunda aparecen, separados por espacios y ordenados cronológicamente, los $N$ años de nacimiento (si en un mismo año nació más de una persona, el año aparecerá repetido).

Detrás del último caso aparece una línea con dos ceros.

## Salida

Para cada caso de prueba el programa escribirá una línea con el mayor número de personas que ha nacido en un periodo de $A$ años. Ten en cuenta que si un periodo de $A$ años comienza en el año $D$ entonces abarca hasta el año $D + A - 1$, inclusive.

## Entrada de ejemplo

```
10 5
1 1 4 4 5 6 6 7 10 11
10 6
1 1 4 4 5 6 6 7 10 11
0 0
```

## Salida de ejemplo

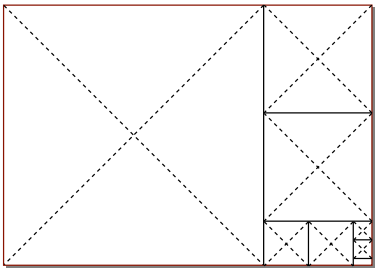
```
6
7
```

# G
# Haciendo pajaritas de papel

Las pajaritas de papel son una de las piezas de *papiroflexia* más conocidas. Partiendo de una porción cuadrada de papel (de igual ancho que alto), y realizando varios dobleces, es posible formar una figura muy reconocible.

Cuando se parte de un trozo de papel que no es cuadrado, lo primero es recortarlo para conseguirlo. Esto ¡alarga la diversión! Es posible seguir troceando la porción sobrante para hacer pajaritas más pequeñas, hasta que la habilidad para manipular trozos minúsculos de papel nos lo permita.

Por ejemplo, si partimos de un DIN-A4, de dimensiones 297×210 milímetros, podremos hacer una pajarita con el cuadrado de 210×210 mm, y nos sobrará un trozo de papel de 87×210. Con él, podremos hacer dos pajaritas con dos cuadrados de 87×87, y nos sobrará un pequeño trozo de 87×36, y así sucesivamente hasta no poder más.



Sabiendo que siempre se intentan hacer las pajaritas lo más grandes posible, y que es imposible hacer pajaritas con trozos menores a 10×10 milímetros, ¿cuántas se pueden hacer para un tamaño inicial de papel?

## Entrada

El programa leerá, de la entrada estándar, múltiples tamaños de papel, cada uno en una línea. El tamaño se especificará con dos números $0 < ancho,alto \leq 10.000.000$. La entrada terminará con una línea con dos 0.

## Salida

Para cada caso de prueba, el programa escribirá el número de pajaritas que se pueden realizar con el trozo de papel de partida, sabiendo que se prefiere hacer menos pajaritas siempre que sean lo más grandes posible. Además, se considera imposible hacer pajaritas con porciones de papel menores a 10×10.

## Entrada de ejemplo

```
297 210
10 10
9 100
0 0
```

## Salida de ejemplo

```
7
1
0
```

# H
# Distancia al siguiente capicúa

Los *números capicúa* son aquellos que, tras ser escritos, se leen igual de izquierda a derecha que de derecha a izquierda. Por ejemplo, los números 11, 474 o 9.889 son capicúa.

Como ocurre con los números primos, entre los números bajos hay muchos números capicúa. Los primeros son el 0, 1, 2, ..., 9, que están todos a distancia 1. Más adelante llegan el 11, 22, 33, ..., 99 que están a distancia 11. El siguiente, sin embargo, vuelve a estar cerca: el 101 está sólo a distancia 2 del anterior.

¿Eres capaz de saber a qué distancia está el siguiente capicúa de un número dado?

## Entrada

La entrada comienza con una línea indicando el número de casos de prueba que vienen a continuación.

Cada caso de prueba es un número mayor o igual que 0 y menor o igual que 2.000.000.000.

## Salida

Para cada caso de prueba se indicará la distancia del número al siguiente capicúa. Ten en cuenta que si el número leído resulta ser capicúa él mismo, habrá que indicar la distancia al siguiente.

## Entrada de ejemplo

```
3
8
27
179
```

## Salida de ejemplo

```
1
6
2
```

# Jan el olvidado

El problema de tener un nombre extraño es que el mundo buscará la forma de no tener que recordarlo incluso aunque hagas algo importante en la vida. Es lo que le ocurrió al polaco Jan Łukasiewicz, creador de lo que se llamó *notación polaca* (o notación *prefija*) por motivos bastante obvios. Consiste en una forma de escribir expresiones (aritméticas, lógicas o algebraicas) en la que se colocan los operadores antes que los operandos. Así, por ejemplo, lo que en la notación tradicional (infija) escribiríamos como $3 + 4$, Jan decidió escribirlo como $+\ 3\ 4$.

Lejos de ser una decisión caprichosa, la principal ventaja de la notación polaca es que no requiere paréntesis para escribir cualquier expresión sin ambigüedad. Esta ventaja es lo suficientemente grande como para que la empresa HP decidiera utilizar una variante, la notación polaca inversa[2], en sus clásicas calculadoras.

| Notación infija | Notación polaca |
|---|---|
| $3 + 4$ | $+\ 3\ 4$ |
| $3 - (4 \times 5)$ | $-\ 3\ \times\ 4\ 5$ |
| $(3 + 4) \times 5$ | $\times\ +\ 3\ 4\ 5$ |
| $(3 - 4) \div (5 + 2)$ | $\div\ -\ 3\ 4\ +\ 5\ 2$ |

## Entrada

El programa leerá, de la entrada estándar, un número indicando la cantidad de casos de prueba que tendrá que procesar. Cada uno, en una línea, contendrá una expresión en notación polaca. Por simplicidad, los números tendrán únicamente un dígito. Los operadores válidos serán suma (`+`), resta (`-`), multiplicación (`*`) y división (`/`). El primer operando se proporciona antes que el segundo, y la división será entera.

## Salida

Para cada caso de prueba el programa deberá escribir, en una línea independiente, el valor de la expresión. Se garantiza que tanto los resultados parciales como los finales serán menores que $2^{31}$ en valor absoluto, y nunca habrá divisiones por 0.

## Entrada de ejemplo

```
4
+ 3 4
- 3 * 4 5
* + 3 4 5
/ - 3 4 + 5 2
```

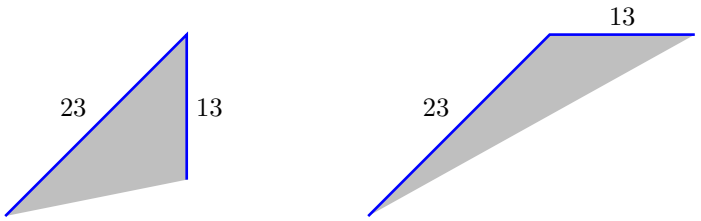
## Salida de ejemplo

```
7
-17
35
0
```

[2]En la notación polaca inversa (también llamada notación *postfija*) el operador se escribe *detrás* de los operandos. El ejemplo del enunciado quedaría por tanto 3 4 +.

# J
# El triángulo de mayor área

A partir de dos segmentos de cualquier longitud, es posible formar un triángulo añadiendo un tercero. De hecho, podremos formar un número infinito de triángulos dependiendo del ángulo que formemos con los dos segmentos originales:




De todos esos triángulos hay uno especial por ser el de *mayor área*. ¿Sabes encontrarlo?

## Entrada

El programa deberá procesar múltiples casos de prueba, cada uno recibido en una línea por la entrada estándar.

Un caso de prueba estará formado por una pareja de números naturales mayores que 0 y menores que 1.000, cada uno indicando la longitud de uno de los segmentos. La entrada terminará con un caso de prueba especial, que no deberá procesarse, en el que ambos números serán 0.

## Salida

Para cada caso de prueba el programa escribirá, en una línea, el área del mayor triángulo que se pueda formar con los dos segmentos. Los decimales de la salida se truncarán, por lo que el resultado escrito siempre será un número entero.

## Entrada de ejemplo

```
23 13
16 29
0 0
```

## Salida de ejemplo

```
149
232
```