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

Programa-Me 2018
Final Nacional
Problemas

Patrocinado por

accenture technology

CICE
LA ESCUELA PROFESIONAL DE NUEVAS TECNOLOGÍAS

eni ediciones

Universidad Complutense Madrid

MÁSTER EN DESARROLLO DE VIDEOJUEGOS

Linux Professional Institute

aws

Ejercicios realizados por

Facultad de Informática
Universidad Complutense de Madrid

Realizado en la **Facultad de Informática (U.C.M.)**
15 de junio de 2018

Universidad Complutense Madrid

15 de junio de 2018
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)
- Yolanda Ortega Mallén (Universidad Complutense de Madrid)
- Alberto Verdejo López (Universidad Complutense de Madrid)

A
Entrando en pelotón

Cuando en una vuelta ciclista los corredores llegan *en pelotón*, todos reciben el mismo tiempo para esa etapa, incluso aunque no hayan llegado exactamente en el mismo instante. Se hace así porque es físicamente imposible, por obvios motivos de espacio, que todos lleguen a la vez. De modo que para evitar accidentes (y, por qué no decirlo, simplificar la gestión), se les asigna a todos el mismo tiempo y problema resuelto.

Ahora eso sí, en cuanto hay *un corte* entre dos ciclistas, se utiliza para el rezagado el tiempo real en el que haya llegado. Si detrás de él llega un segundo *pelotón*, todos ellos tendrán el mismo tiempo, otra vez.

Entrada

El programa deberá leer, por cada caso de prueba, un primer número $1 \leq n \leq 100$ indicando el número de corredores que han conseguido terminar la etapa.

A continuación, vendrán n líneas con los tiempos de esos corredores en formato `HH:MM:SS`, indicando las horas, minutos y segundos reales que ha tardado cada uno en llegar a la meta (nunca más de 24 horas). Aunque los nombres de los corredores no se proporcionan, los tiempos están ordenados por sus apellidos, por lo que, en la práctica, tienen un orden arbitrario.

La entrada termina con un `0`.

Salida

Por cada caso de prueba se escribirá la posición de llegada de cada corredor, en el mismo orden en el que se recibieron en la entrada. Todos los corredores que son calificados con el mismo tiempo de llegada (tras la agrupación por el pelotón) deben recibir el mismo puesto. Se considera que se ha producido *un corte* de corredores si su distancia en tiempo es mayor que un segundo.

Después de cada caso de prueba el programa escribirá una línea en blanco.

Entrada de ejemplo

```
3
01:01:01
01:00:59
01:01:00
4
01:03:00
01:01:00
01:00:59
01:01:00
0
```

Salida de ejemplo

```
1
1
1

4
1
1
1

```

B
El cuello de los pilotos

Cualquiera con dos dedos de frente entiende que los ciclistas deben cuidar sobre todo sus rodillas y los tenistas el codo. Lo que es menos conocido es la importancia del cuello en los pilotos de Fórmula 1.

La necesidad de un cuello en forma en estos conductores se debe a las fuertes aceleraciones y deceleraciones que sufren durante las carreras, especialmente en las curvas.

Por eso cuando se incorpora un nuevo circuito en el calendario del mundial sus curvas se analizan minuciosamente. Aunque en un estudio real se miden muchos más factores, nosotros nos conformaremos con contar el número de curvas a la izquierda y a la derecha que tienen esos nuevos circuitos.

Entrada

La entrada está compuesta por distintos casos de prueba, cada uno representando el mapa de un circuito cuyas curvas hay que contar.

La descripión de cada mapa comienza con una línea con dos números *tx* y *ty* que indican su ancho y alto (un mínimo de 3 y un máximo de 100 unidades por cada dimensión).

A continuación aparecen *ty* líneas, con *tx* caracteres cada una. Cada carácter puede ser un punto (“.”) que indica *campo* y una almohadilla (“#”) que indica una sección del circuito. Dentro de éste, el carácter “O” (*o* mayúscula) marca la posición desde la que comienza la carrera, que nunca estará situada en una curva.

En nuestros circuitos los coches siempre van en horizontal o vertical (nunca en diagonal) y recorren el circuito en sentido de las agujas del reloj.

La entrada termina con una línea con dos ceros que no debe ser procesada.

Salida

Por cada caso de prueba se escribirá una única línea con dos números separados por un espacio. El primero indica las curvas hacia la izquierda y el segundo las curvas hacia la derecha que deben hacer los pilotos en el circuito.

Entrada de ejemplo

```
3 3
#O#
#.#
###
8 7
..#####.
..#...#.
###...#.
O.....#.
###...#.
..#####.
........
0 0
```

Salida de ejemplo

```
0 4
2 6
```

C

Multiplicando mal

Después de haber aprendido a sumar y restar (¡incluso llevando!) hace un par de años, Celia no sale de su asombro con la multiplicación. Su profesor insiste en que tiene que multiplicar el primer número por cada uno de los dígitos del segundo, por separado, y luego sumar todos los resultados:

```
   123
 × 123
-----
   369
+ 246
  123
-----
 15129
```

"¡Pero eso es una lata!" — grita amargamente — "¡Hay que dar un montón de pasos, con lo cómodo que era sumar y restar!" Demostrando la rebeldía que tanto temen sus padres cuando piensan en la futura adolescencia, ha decidido que prefiere el método "tradicional" de hacer las cuentas, y multiplica como si estuviera sumando, utilizando de manera independiente los números de las columnas.

```
   123
 × 123
-----
   149
```

Por más que la dicen que así no se hace y que los resultados están mal, prefiere ignorar las críticas y asegura que su método es mucho más rápido y práctico. No obstante, secretamente, reconoce que a veces le dan resultados un poco raros con los ceros de la izquierda, cuando los números tienen diferente longitud.

Entrada

La entrada comenzará con un número con la cantidad de casos de prueba que deberán procesarse. Cada uno está compuesto por una pareja de números, separados por un espacio, que deben multiplicarse. Como Celia no quiere hacer caso, su profesor la castiga poniéndole números muy largos (de hasta 100 cifras) para que, al menos, practique las tablas de multiplicar.

Salida

Para cada caso de prueba el programa escribirá el resultado de la multiplicación al estilo de Celia, incluyendo los posibles ceros a la izquierda.

Entrada de ejemplo

```
4
123 123
423 424
12 123
123 82
```

Salida de ejemplo

```
149
1652
026
166
```

D
Polisílaba es polisílaba

La palabra "*polisílaba*" es polisílaba ya que tiene más de tres sílabas. Este curioso fenómeno de una palabra describiéndose a sí misma también se da en otros vocablos que describen propiedades de las palabras. Por ejemplo, tenemos que "*esdrújula*" es esdrújula y que "*llana*" es llana.

Si vamos a palabras con más de un significado, el mismo esquema de frase aparentemente tautológica podemos tenerlo por ejemplo al afirmar que la naranja es naranja.

Sea como sea, si alguien no sabe el significado de *polisílaba* o de *esdrújula* las frases anteriores serán, efectivamente, simples tautologías vacías de contenido.

Entrada

La entrada comienza con una línea con un número indicando cuántos casos hay que procesar.

Cada caso tiene una frase con el esquema "`X es Y`" en la que tanto `X` como `Y` son palabras sin espacios con hasta 20 caracteres del alfabeto inglés.

Salida

Por cada caso de prueba se escribirá `TAUTOLOGIA` si la frase es una tautología (sin distinguir entre mayúsculas y minúsculas) y `NO TAUTOLOGIA` en caso contrario.

Entrada de ejemplo

```
3
Polisilaba es polisilaba
NARANJA es naranja
Mandarina es naranja
```

Salida de ejemplo

```
TAUTOLOGIA
TAUTOLOGIA
NO TAUTOLOGIA
```

E
El mejor peor día de tu vida

Dicen que el día de tu boda es el mejor momento de tu vida. Eso puede interpretarse como que a partir de ese momento no se levanta cabeza. Si lo ponemos con números significa que podemos representar cada día de nuestra vida con una cifra y el máximo de la serie numérica de tu vida estaría precisamente en ese día.

Si damos por cierto el dicho anterior, los pesimistas no se casarán nunca. A fin de cuentas, sería imposible que levantaran el ánimo sabiendo que nunca nada superará esas fugaces horas en las que te comprometiste para siempre.

Los optimistas, no obstante, no ven las cosas de forma tan horrible. Que el día de tu boda sea el mejor no significa que luego las cosas vayan decayendo continuamente. Muy al contrario, habrá días malos a los que seguirán días buenos que se podrán disfrutar. Es cierto que nunca superarán el máximo alcanzado el día de tu enlace matrimonial, pero no por eso hay que dejar de valorarlos.

Esos optimistas empedernidos encuentran un motivo de alegría incluso los días más horribles. Se conforman con pensar que cuanto peor es un día, mayor será la diferencia con el mejor día que aún les queda por vivir, y el contraste hará que cuando llegue lo disfruten más.

Entrada

La entrada estará compuesta por distintos casos de prueba, cada uno representando la serie de números que condensan la vida de una persona.

Cada caso de prueba ocupa dos líneas. La primera contiene un único número N con el número de días para el que hay valoración ($2 \leq N \leq 200.000$). A continuación aparecerá una segunda línea con las valoraciones de esos N días, que estarán siempre entre 0 y 10^9.

Salida

Para cada caso de prueba se escribirá una única línea con un número, indicando el valor que los optimistas dan al mejor peor día de su vida. El valor dado a cada día es la diferencia entre la valoración del mejor día que aún queda por vivir (incluído él) menos su propio valor.

Entrada de ejemplo

```
3
1 4 2
3
2 4 1
3
4 2 1
```

Salida de ejemplo

```
3
2
0
```

F
Pesando patatas

Cuando antiguamente se utilizaban las balanzas de dos platos para pesar, los comerciantes llevaban una colección de pesas que colocaban junto con la mercancía en los platos hasta que la balanza quedaba equilibrada. De esta forma, averiguaban la cantidad exacta que estaban vendiendo y emitían el precio.

La colección de pesas exacta dependía de la destreza del vendedor. Hoy día, habituados como estamos a la *base 2*, es fácil entender que las pesas podrían tener pesos potencia de dos. En ese caso basta con colocar en un plato la mercancía y en el otro aquellas pesas cuyos bits estén a uno en la representación en binario del número.

Sin embargo si permitimos que mercancía y pesas compartan el mismo plato, las posibilidades se disparan. Por ejemplo, con una pesa de 1 Kg y otra de 100 gr, podremos averiguar si una colección de patatas pesa 900 gr. Basta con colocar en un platillo la pesa grande y en el otro la pesa pequeña y las patatas y mirar si la balanza queda equilibrada.

Una forma óptima (en cuanto a número de pesas distintas necesarias) para pesar es utilizar esta idea con pesas potencia de 3. Lo difícil es saber rápidamente cómo colocarlas...

Entrada

La entrada estará compuesta por varios casos de prueba, cada uno en una línea.

Cada caso de prueba será un único número que indicará la cantidad de patatas, en gramos, que queremos pesar. El peso no excederá los 10^9 gramos.

Al último caso de prueba le sigue una línea con un 0, que no debe procesarse.

Salida

Para cada caso de prueba se escribirá, en una línea independiente, la configuración que debemos tener en la balanza.

La línea comenzará con los pesos (en orden creciente) de las pesas del platillo en el que *no* tenemos la mercancía, separados por espacio. A continuación le seguirán los caracteres `==X==` simbolizando la parte central de la balanza. Después se indicarán, también en orden creciente y separados por un espacio, las pesas a colocar en el segundo platillo, seguidos de la palabra `patatas`.

Las pesas disponibles tienen siempre pesos en gramos potencia de tres, y tan altos como necesites. Pero *sólo tienes una pesa de cada peso.*

Entrada de ejemplo

```
1
5
8
79
0
```

Salida de ejemplo

```
1 ==X== patatas
9 ==X== 1 3 patatas
9 ==X== 1 patatas
1 81 ==X== 3 patatas
```

G
Montando semáforos

Shan Cho Kao es un empresario chino famoso por sus fábricas de semáforos. Todas ellas tienen una línea de producción que comienza con un operario que va colocando por orden las bombillas de los tres famosos colores: rojo, amarillo, verde; rojo, amarillo, verde... Más adelante, otros trabajadores van cogiendo de la línea las bombillas, y las ensamblan en los semáforos, que acabarán en las ciudades de medio mundo.

Todas las fábricas han funcionado perfectamente durante años, hasta que hace unos días, en la sede de Lan Li Hao cometieron el error de contratar a Confun Dio para colocar las luces al principio de la línea, sin que éste se atreviera a reconocer que es daltónico y no distingue los colores.

El resultado es que las luces están llegando a los ensambladores completamente desordenadas. Éstos, entrenados durante años a coger las bombillas consecutivas, no son capaces de reaccionar y adaptarse, por lo que se montan unos atascos en la línea de producción que no dicen nada positivo sobre el producto que fabrican.

En ocasiones, el azar hace que las bombillas puedan retirarse correctamente aprovechando que la retirada de tres luces hace que otras dos queden consecutivas aunque no lo estuvieran en un principio. Por ejemplo si Confun Dio coloca las luces en la secuencia rojo, amarillo, rojo, amarillo, verde, verde, se pueden construir dos semáforos extrayendo las luces de la línea de producción. Si representamos los colores por sus iniciales:

$$\texttt{RA}\textit{RAV}\texttt{V} \Rightarrow \texttt{RAV} \Rightarrow \emptyset$$

Los operarios cogen siempre luces consecutivas, que deben estar en el orden rojo, amarillo, verde. Sabiendo esto, dada una secuencia de luces, ¿cuántos semáforos como máximo se pueden construir?

Entrada

La entrada consiste en múltiples líneas, cada una de ellas con la secuencia de colores de las bombillas tal y como las ha colocado Confun Dio en un día de trabajo. Siempre se especifican como una cadena con las letras `R`, `A` o `V`, para cada uno de los colores rojo, amarillo y verde. El número máximo de bombillas por día que es capaz de poner es 500.000.

Salida

Por cada caso de prueba el programa escribirá una única línea con el número máximo de semáforos completos y correctos que pueden construirse con los operarios actuales.

Entrada de ejemplo

```
RARAVV
RAV
VAR
RAVV
```

Salida de ejemplo

```
2
1
0
1
```

H
Buscando el pinchazo

Hacer rutas en bici está muy bien... hasta que pinchas. No es solo el engorro de tener que cambiar la cámara de la rueda en mitad de cualquier camino, es que luego al llegar a casa toca arreglar el pinchazo.

Eso supone llenar un barreño con agua, hinchar la cámara pinchada y sumergirla en agua, sección a sección, hasta ver aparecer las burbujillas delatoras que te dicen el punto que ha amargado tu paseo.

Y, por si pinchar no es suficiente mala suerte, la búsqueda del pinchazo en la cámara vuelve a poner a prueba tu relación con la diosa Fortuna. El primer punto de la cámara que sumerges en agua con la esperanza de encontrar el pinchazo y el sentido en el que vayas girando la cámara para sumergir los tramos siguientes afectan, y mucho, al tiempo que tardarás en ver las esperadas burbujas. Si todo va bien, encontrarás el pinchazo antes de recorrer la mitad de la cámara. Si la mala suerte te ha acompañado hasta casa, las burbujas aparecerán después de haber sumergido la cámara prácticamente entera.

Entrada

La entrada comienza con un número que indica cuántos casos de prueba habrá que procesar.

Cada caso de prueba son dos números $0 \leq i, p < 360$ indicando el punto inicial de la cámara donde empiezas a buscar el pinchazo (el primero que sumerges) y la posición del pinchazo, respectivamente. Ambos son números enteros indicando *el grado* en la circunferencia de la cámara.

Salida

Por cada caso de prueba se escribirá "`ASCENDENTE`" si es mejor realizar la búsqueda del pinchazo girando de manera ascendente en grados desde el punto de partida, y "`DESCENDENTE`" si es mejor ir en sentido opuesto. Si el pinchazo está en el punto de inicio o en el opuesto de la circunferencia se escribirá "`DA IGUAL`".

Ten en cuenta que, al ser grados de una circunferencia, la posición `0` es la misma que la `360`.

Entrada de ejemplo

```
3
90 91
0 359
0 180
```

Salida de ejemplo

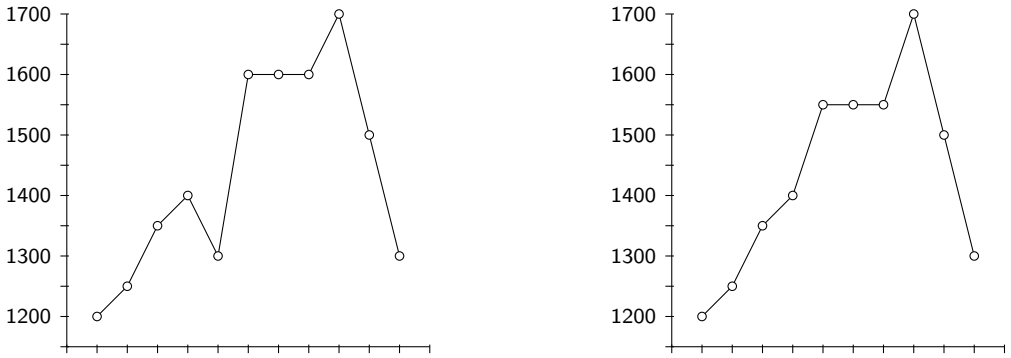
```
ASCENDENTE
DESCENDENTE
DA IGUAL
```

Caminando voy

Con la llegada del buen tiempo, el grupo de senderistas *Caminando voy* quiere preparar una serie de excursiones por la Sierra de Madrid. Las excursiones deben ser aptas para todos los socios (hay un numeroso grupo de jubilados y otro de niños de corta edad).

En concreto, se requiere que las cuestas arriba no se hagan demasiado penosas. Para ello, para cada excursión se ha confeccionado un perfil de desniveles, consistente en una secuencia de cotas de altura (valores enteros no negativos). Las cuestas arriba se corresponden con segmentos estrictamente crecientes, y el desnivel de una cuesta será la diferencia entre su cota más alta y su cota más baja. Una excursión se considerará *apta* si en todas las cuestas arriba el desnivel no supera un cierto valor estipulado $D \geq 0$.

Por ejemplo, suponiendo $D = 300$, una excursión con el perfil de la izquierda sería apta, mientras que una con el perfil de la derecha no.



Ahora necesitan un programa que clasifique todas las excursiones que tienen programadas. ¿Puedes ayudarles?

Entrada

La entrada consta de una serie de casos de prueba, cada uno correspondiente a una excursión. Cada caso ocupa dos líneas. En la primera aparecen dos números separados por un espacio: el desnivel máximo permitido D (entre 0 y 1.000.000) y el número N (entre 1 y 200.000) de cotas de altura que forman la excursión. En la segunda línea aparecen las N cotas (números entre 0 y 1.000.000), separadas por espacios.

Salida

Para cada caso de prueba, el programa escribirá `APTA` si la excursión es apta para el desnivel máximo permitido, y `NO APTA` en caso contrario.

Entrada de ejemplo

```
300 11
1200 1250 1350 1400 1300 1600 1600 1600 1700 1500 1300
300 10
1200 1250 1350 1400 1550 1550 1550 1700 1500 1300
0 4
10 10 10 10
```

Salida de ejemplo

```
APTA
NO APTA
APTA
```

J
Al mundial en transatlántico

El 4 de mayo de 1949 tuvo lugar la *tragedia de Superga*, un accidente aéreo en el que el avión donde viajaba todo el equipo de fútbol del *Torino Football Club* se estrelló, dejando 31 víctimas mortales, incluídos los 18 jugadores del equipo.

El *Torino* estaba viviendo los mejores momentos de su historia, con cinco campeonatos de liga consecutivos, por lo que 10 de los 11 jugadores titulares de la selección italiana de aquel entonces eran suyos. El accidente conmocionó no solo a Italia, sino a toda Europa.

Cuando al año siguiente el equipo de la selección italiana asistió al mundial de fútbol de Brasil (el del famoso *Maracanazo*), conmocionado aún por el accidente decidió hacer el viaje en transatlántico. Con un equipo diezmado por la tragedia y cansado por más de dos semanas de viaje, su actuación fue bastante tibia. Además, el plan era que entrenaran durante el trayecto en la cubierta del barco, pero pronto se quedaron sin balones, que terminaron en el agua.

Cuenta la leyenda que una tarde, cuando solo les quedaban 20 balones, el delantero principal decidió practicar el tiro de faltas. Colocaba los balones en orden, tiraba a gol, los recogía, volvía a colocarlos por orden, y empezaba de nuevo.

El problema fue que uno de cada diez disparos acababa con el balón en el agua. Antes de la hora de la cena, se había quedado sin ninguno.

Entrada

Cada caso de prueba consta de dos números, $1 \leq b$, $p \leq 100$, indicando, respectivamente, el número de balones inicial, y tras cuántos tiros acaba un balón en el agua. Por ejemplo, si p es 10, significa que se pierde un balón de cada 10 disparos: nuestro delantero lanza los nueve primeros bien, y el décimo acaba en las aguas del Atlántico.

La entrada termina con dos ceros.

Salida

Por cada caso de prueba el programa escribirá el número del balón (entre 1 y b) que acaba en el agua en último lugar.

Entrada de ejemplo

```
1 10
2 1
2 2
10 10
0 0
```

Salida de ejemplo

```
1
2
1
8
```