Concurso de programación

Las 12 uvas

2022

Problemas


Premios patrocinados por

Ejercicios realizados por

Facultad de Informática
Universidad Complutense de Madrid

Organizado desde la **Facultad de Informática (UCM)**
31 de diciembre de 2022

Universidad Complutense Madrid

31 de diciembre de 2022
https://las12uvas.es

Listado de problemas

Autores de los problemas:

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

Revisor:

- Alberto Verdejo López (Universidad Complutense de Madrid)

A
Contando desde el cero

Cuando los niños aprenden a contar, siempre empiezan su cantinela con el número "uno". Hacerlo empezando en 0 no tiene sentido porque el mero hecho de contar asume que hay algo para ser contado y, por tanto, al menos habrá 1. Esto es así hasta tal punto que los *números naturales* de los matemáticos *no incluían el 0* y, de hecho, dependiendo del ámbito de uso, en algunos contextos aún se asume que no lo incluyen.

Y es que aunque hoy pueda resultar sorprendente, el concepto del *cero* ha sido históricamente complejo y tardó mucho en asentarse. Aunque apareció tímidamente con anterioridad, a Europa el *cero posicional* llegó gracias a Fibonacci en el siglo XII con su libro *Liber abaci* ("El libro del ábaco") donde introducía el álgebra árabe (copiada, a su vez, de los indios).

Anteriormente, varias civilizaciones antiguas disponían de símbolos para representarlo (el Antiguo Egipto, Babilonia, la Antigua Grecia y la civilización Maya, por ejemplo) pero no aprovecharon su verdadero potencial. Por su parte, los romanos, que fueron grandes legisladores y arquitectos, fueron malos matemáticos, y en los números romanos no había forma de representar el 0.

Ante esta situación, no es de extrañar que no existiera el año 0. El año 1 antes de Cristo fue seguido por el año 1 después de Cristo. En realidad, quien impulsó este modo de contar los años fue Beda, un monje benedictino nacido en el siglo VII que sí intuía la existencia del cero. Pero, pensó, si no hay "mes cero", ni "día de la semana cero", ni "siglo cero", ¿por qué ha de haber "año cero"?

Entrada

La entrada comienza con un número indicando cuántos casos de prueba deberán ser procesados. Cada uno es un número entre –3.000 y 3000 representando un año. Los valores negativos indican años antes de Cristo y los positivos después de Cristo (o de la "era común"). Al no haber existido, en la entrada no habrá nunca un 0.

Salida

Por cada caso de prueba, el programa escribirá el año correspondiente en un supuesto calendario en el que *sí* existió el año 0, es decir un calendario donde el año 1 habría sido el año 0, manteniendo los anteriores igual.

Entrada de ejemplo

```
3
2023
1
-100
```

Salida de ejemplo

```
2022
0
-100
```

B
Gálibo

Se conoce como el gálibo a la dimensión máxima que debe tener un vehículo para poder pasar por una zona de la vía de tamaño reducido como un túnel o un puente. Existen distintos tipos de gálibo y de formas de calcularlo que dependen del tipo de vehículo. Por ejemplo en los barcos el gálibo de un puente se mide teniendo en cuenta el nivel de la crecida más grande conocida. En trenes, el gálibo debe considerar no solo el alto sino también los anchos a distintas alturas.

En carretera el gálibo del que hay que preocuparse normalmente suele ser el vertical. En este caso la altura libre por debajo del puente debe tener en cuenta la altura máxima de los vehículos y un margen de seguridad relacionado con la amortiguación y cómo esta absorbe los movimientos verticales, así como cierta tolerancia por posibles imprecisiones a la hora de construir la estructura y por futuras operaciones de reasfaltado de la vía. Se puede incluso considerar, además, un margen de confort para que los conductores de los vehículos pesados se sientan cómodos (que dependerá de cómo se aproxime la vía al puente).

Para evitar problemas de gálibo para los vehículos más grandes, en algunas carreteras se colocan puentes de calibración que permiten a los conductores averiguar con tiempo suficiente si un puente próximo no tiene la altura mínima necesaria.

En ausencia de estos puentes de calibración, se puede confiar en la información que aparece en el propio puente. En algunos países hay pequeños carteles en el frontal del puente justo debajo de cada uno de los carriles para que los conductores puedan conocer su altura en cada uno de ellos. Gracias a estos carteles si se quiere viajar seguro se puede hacer un primer trayecto en un vehículo más pequeño registrando esas alturas para conocer si el vehículo grande podrá pasar por debajo de todos ellos.

Entrada

La entrada está compuesta de un número indeterminado de casos de prueba.

Cada caso de prueba comienza con una línea con un único número indicando el número de puentes (hasta 100) que hay en el trayecto que se desea hacer. Tras eso vendrá una línea por cada puente con su descripción. Esta comienza con el número de carriles que pasan por debajo del puente en el sentido de nuestro trayecto (como mucho 5) seguido de la altura del puente en cada uno de ellos en centímetros (un número entre 1 y 800).

Tras el último caso de prueba viene una línea con un `0` que no debe procesarse.

Salida

Por cada caso de prueba se escribirá la altura máxima que puede tener el camión para poder realizar el trayecto, teniendo en cuenta que este puede utilizar cualquiera de los carriles disponibles.

Entrada de ejemplo

```
2
3 300 300 250
2 325 300
2
1 450
1 400
0
```

Salida de ejemplo

```
300
400
```

C
SemVer++

El *versionado semántico* (SemVer, por *Semantic Versioning*) es un esquema de numeración de versiones de software, muy utilizado especialmente en bibliotecas por reducir la pesadilla de las dependencias (*dependency hell*). El número de versión tiene relación directa con la *compatibilidad del API*, de modo que es fácil saber, de manera automática, si una determinada aplicación que depende de una versión concreta de una biblioteca podrá funcionar con una versión posterior. El número de versión tiene por tanto un *significado*, de ahí el nombre de *versionado semántico*.

Bajo este esquema, la versión de una biblioteca se compone de tres números separados por punto, cada uno con un significado, que normalmente se representan como *Major.Minor.Patch*. Así, por ejemplo, la versión 3.1.4 tiene como versión principal la 3, subversión 1 y revisión (o parche) 4.

Un incremento en el número principal de la versión (*major*) indica *un cambio incompatible de API*. Eso significa que, por ejemplo, una aplicación que utilice la versión 3.1.4 de una biblioteca no podrá ser compilada con la versión 4.0.0 porque la forma en la que funciona ha cambiado completamente.

Un incremento en el número de subversión (*minor*) indica una mejora de la librería, ampliando su funcionalidad pero no cambiando nada de lo que ya tuviera la versión previa. Por su parte, un cambio en el número de revisión (*patch*) indica un arreglo de algún error descubierto de la versión previa, pero sin cambios en la funcionalidad.

El problema del versionado semántico es que es imposible saber qué versión seguirá a una dada, porque podría cambiar cualquiera de los tres números. Eso sí, al menos sabemos que cuando un número cambia, los siguientes se reinician a 0.

Entrada

La entrada comienza con un primer número indicando cuántos casos de prueba deberán ser procesados. Cada uno está compuesto por dos números de versión de una determinada biblioteca software, en formato *SemVer*. Todos los números estarán entre 0 y 1000.

Salida

Por cada caso de prueba el programa escribirá "`SI`" si el segundo número de versión *podría ser* el siguiente al primero, y "`NO`" en caso contrario.

Entrada de ejemplo

```
3
3.1.4 3.1.6
0.0.10 1.0.0
1.2.3 1.3.1
```

Salida de ejemplo

```
NO
SI
NO
```

D
Yo lo coloco

El *calambur* es un artificio lingüístico utilizado de vez en cuando por escritores y humoristas para provocar la sonrisa de los lectores o espectadores. Consiste en aprovechar que una misma frase puede tener dos significados dependiendo de cómo se agrupen las sílabas pronunciadas.

Un ejemplo muy conocido es la frase "Yo lo coloco y ella ¡lo quita!" que, estableciendo la separación entre palabras de forma distinta puede interpretarse como "Yo loco, loco. Y ella... loquita".

Es posible, eso sí, que el calambur más famoso de la lengua española sea uno atribuido a Quevedo que aprovechó el recurso fonético para llamar coja a la reina doña Isabel de Borbón. Parece ser que se presentó ante la reina con un ramo de flores en cada mano y dijo "Entre el clavel blanco y la rosa roja, su majestad escoja".

El calambur suele basarse en la forma en la que las palabras suenen por lo que en ocasiones las letras en ambas interpretaciones no coinciden. Por ejemplo ocurre con "¿Por qué lavó la rueda?" frente a "¿Por qué la bola rueda?". Otras veces (las menos) las dos interpretaciones tienen exactamente las mismas letras pero al leerlos suenan de forma muy distinta, como "Servil, letal, impía" y "Servilleta límpia".

¿Eres capaz de ver si dos frases forman un calambur? Para hacerlo sencillo, solo consideraremos el último grupo: aquel que, aunque la pronunciación sea muy distinta, tenga exactamente las mismas letras.

Entrada

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

Por cada caso de prueba aparecerán dos líneas, una con cada frase. Las líneas estarán formadas exclusivamente por letras del alfabeto inglés (no habrá eñes ni vocales acentuadas, por ejemplo), espacios y signos de puntuación (puntos, comas, dos puntos o punto y coma).

Cada línea tendrá un máximo de 500.000 caracteres.

Salida

Por cada caso de prueba se escribirá una línea con un `SI` si a la vista de las dos líneas forman un calambur (con exactamente las mismas letras) y `NO` en caso contrario. Dos frases exactamente iguales las consideraremos también calambur.

Entrada de ejemplo

```
2
Yo lo coloco y ella lo quita.
Yo loco, loco. Y ella... loquita.
El pan esta blando
El pan esta hablando
```

Salida de ejemplo

```
SI
NO
```

E
Pipos

Pese a haber nacido en martes y 13, Jose Mayus T. es un supersticioso. No le parece suficiente comerse las 12 uvas religiosamente al son de las campanadas de fin de año para evitar que una maldición caiga sobre su cabeza. También se complica la vida pensando que las uvas que se coma deben tener, exactamente, el número de pipos que digan los dígitos del nuevo año, y en el mismo orden.

Empezó a hacerlo cuando aún era joven, al pasar al año 2000. Se dio cuenta de que la primera uva que se comió tenía dos semillitas, y las tres siguientes no tuvieron ninguna, de modo que los pipos formaron, de casualidad, el número 2000. Lo sorprendente es que le pasó lo mismo con las 4 uvas siguientes y con las cuatro últimas.

Para Jose, el año 2000 fue muy bueno y, achacándolo a aquella coincidencia, desde entonces todas las nocheviejas se organiza para conseguir que sus 12 uvas formen 3 grupos de 4, y en cada grupo el número de pipos de las uvas formen los dígitos del nuevo año.

La edad le está volviendo cada vez más fetichista y este año, que organiza la cena de nochevieja para la familia, quiere que todos los asistentes se coman las uvas siguiendo su excéntrica manía. Tiene un montón de uvas y, con paciencia infinita, ha averiguado, mirándolas una a una al trasluz, cuantos pipos tiene cada una. Ahora necesita saber a cuánta gente, como máximo, podrá darle las uvas correctas.

Entrada

El programa deberá leer, de la entrada estándar, un conjunto de casos de prueba. Cada uno comienza con un número indicando el año que está a punto de empezar (siempre de cuatro dígitos). A continuación van, en otra línea, diez números menores que 10^9 indicando cuántas uvas no tienen ningún pipo, cuántas tienen 1, y así sucesivamente hasta 9 pipos. Nunca ha encontrado ninguna que tenga más.

La entrada termina con un 0, que no debe procesarse.

Salida

Por cada caso de prueba el programa deberá escribir el número de personas a las que Jose podrá dar las uvas de modo que se puedan formar, con sus pipos, tres veces los dígitos del nuevo año.

Entrada de ejemplo

```
2023
6 6 6 6 6 6 6 6 6 6
2031
30 30 30 30 30 30 30 30 30 30
3791
30 5 7 8 3 10 2 7 0 13
0
```

Salida de ejemplo

```
1
10
1
```

F
Restanacci

La sucesión de Fibonacci es archiconocida entre los estudiantes de informática. Es una serie infinita de números, que comienza con un 0 y con un 1, y a partir de ahí cada valor de la secuencia se construye sumando los dos valores inmediatamente anteriores.

Originalmente propuesta como una solución a un problema de *la cría de conejos*, la serie tiene infinidad de propiedades que los matemáticos se han entretenido en buscar y detallar. Para los informáticos, sin embargo, es un ejemplo de *recusión doble*.

El problema que tiene esta sucesión, a efectos prácticos, es que crece bastante deprisa. Aunque el primer número de Fibonacci (fib_0) es 0 y el segundo (fib_1) es 1, luego el valor se dispara y, por ejemplo, fib_{46} es 1.836.311.903. El siguiente valor de la secuencia ya no entra en un entero de 32 bits con signo, el tipo habitual en muchos lenguajes de programación.

Eso quita mucha diversión porque se puede subir muy poco... ¡hasta hoy! Para solucionar este problema, hemos inventado *la sucesión de Restanacci* donde cada valor se calcula como *la resta* de los dos anteriores. Además, para añadir un poco de variedad, los dos primeros valores de la secuencia pueden ser cualquiera. ¡Eso aumenta todavía más la diversión!

- $res_0 = a$
- $res_1 = b$
- $res_n = res_{n-1} - res_{n-2}$

Entrada

Cada línea de la entrada estándar contendrá un caso de prueba que el programa tendrá que procesar. Un caso de prueba son tres números naturales, $0 \leq a, b, n \leq 10^9$. La entrada termina con tres ceros, que no deben procesarse.

Salida

Por cada caso de prueba el programa escribirá el valor de res_n, sabiendo que $res_0 = a$ y $res_1 = b$.

Se garantiza que en ningún caso el valor de la sucesión será mayor, en valor absoluto, a 2×10^9.

Entrada de ejemplo

```
0 1 0
2 3 1
7 4 2
123456 654321 14
0 0 0
```

Salida de ejemplo

```
0
3
-3
530865
```

G
Lámpara de pared

Tras verla en una película, un fabricante de lámparas quiso imitar una lámpara de pared que aparecía en la casa de los protagonistas. El diseño no era especialmente bonito pero tampoco era feo y, sobre todo, le permitiría deshacerse de un montón de tiras de metal que tenía en el almacén.

Además del armazón donde va la bombilla y del cristal que la cubre, el diseño consta de dos varillas de metal horizontales de la misma longitud colocadas en la zona inferior y dos varillas, también iguales, a cada uno de los lados. Eso significa que, para cada lámpara, necesita dos varillas del mismo tamaño por un lado y cuatro por otro.

En el almacén tiene numerosas piezas de metal que podrían servir pero que no puede recortar. Si sus longitudes concretas no importan (el tamaño del armazón lo puede ajustar a voluntad y no le importa que la lámpara sea más alta que ancha, más ancha que alta o incluso cuadrada), ¿cuántas lámparas podrá hacer en total?

Entrada

La entrada está compuesta por distintos casos de prueba, cada uno ocupando dos líneas.

La primera línea de cada caso contiene el número de varillas que tiene en el almacén (hasta 1000). La segunda línea tiene un número por cada segmento con su longitud en centímetros (un número entre 1 y 100).

El último caso de prueba viene seguido de una línea con un `0` que no debe procesarse.

Salida

Por cada caso de prueba se escribirá una única línea con el número máximo de lámparas que pueden fabricarse.

Entrada de ejemplo

```
6
2 2 2 2 2 2
7
2 2 1 2 1 2 2
8
1 2 3 1 2 3 1 2
0
```

Salida de ejemplo

```
1
1
0
```

H
Atascos con público

La M-30 es una vía de circunvalación de unos 30 kilómetros de longitud que rodea el centro de Madrid. Se comenzó a construir en 1970 y desde entonces da servicio a miles (e incluso cientos de miles) de vehículos diarios. Tanto es así que sus atascos son famosos y nunca falta en la información del tráfico que todas las mañanas dan en las horas punta las radios y televisiones.

En 2006 se comenzó una inmensa obra para soterrar un buen número de kilómetros de la vía liberando miles de metros cuadrados de superficie que pasaron a ser en su mayor parte zonas verdes. En ese tramo soterrado se tuvo que dejar, eso sí, una pequeña parte que seguía siendo de superficie pues discurría por debajo de las gradas de un estadio de fútbol (Vicente Calderón) y era imposible soterrar esa zona teniendo una construcción tan grande encima.

Eso cambió cuando el club de fútbol propietario del estadio se mudó a otro en las afueras. En 2019 comenzaron los trabajos de demolición y el trazado de la M-30 se vio alterado durante las obras. Curiosamente durante una de las fases, las gradas del estadio seguían en pie y la M-30 pasaba justo delante, dando la sensación de que se habían colocado unas majestuosas gradas para que los famosos atascos pudieran tener espectadores.

El alcalde de una ciudad al sur de Madrid llamada Le-ganéh vio fotos de la M-30 con las gradas y, haciendo honor al nombre de la ciudad, quiso superar al alcalde de Madrid poniendo gradas en su carretera principal y cobrar entrada a todo aquel que quisiera ver sus atascos. Dividió la carretera en tramos e hizo una estimación de las ganancias esperadas en cada uno para decidir en cuáles las colocaría. Eso sí, teniendo en cuenta que entre grada y grada debía de haber al menos k tramos sin público para no despistar mucho a los conductores.

Afortunadamente un asesor le dijo a tiempo que lo de la grada de la M-30 era temporal y el proyecto en Le-ganéh se canceló. Aun así queremos ver si la planificación de dónde poner las gradas estaba bien hecha o no.

Entrada

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

Cada caso de prueba tiene dos líneas. En la primera aparecen dos números con el número de tramos (hasta 100.000) y el número de tramos vacíos que hay que dejar como mínimo entre dos gradas. La segunda línea contiene tantos números como tramos con las ganancias que podrían obtenerse en cada uno de ellos en caso de colocarse ahí una grada (hasta 10^9).

Salida

Por cada caso de prueba se escribirá el beneficio máximo que se puede conseguir teniendo en cuenta que se pueden colocar gradas en todos los tramos que se quiera siempre y cuando entre dos tramos con gradas se respete el mínimo número de tramos sin ellas.

Entrada de ejemplo

```
3
3 1
600 1000 600
3 1
600 2000 600
6 1
1000000000 60 10 1000000000 10 1000000000
```

Salida de ejemplo

```
1200
2000
3000000000
```

I
Abuelos felices

Han ido pasando los años y la familia de mis abuelos no ha dejado de crecer. Lo que comenzó siendo una joven pareja recién llegada a la ciudad desde un pequeño pueblecito se ha convertido en una gran familia con un montón de hijos que a su vez se han casado y han tenido su propia descendencia. Y ahora los primos vamos teniendo nuestras parejas lo que hace que seamos todavía más.

Esa gran familia es una gozada en verano cuando organizamos un día de campo en el que toda la tropa, mesas de picnic en mano, nos vemos en algún encinar y disfrutamos de una barbacoa y de juegos al aire libre.

En navidades las cosas son un poco más difíciles porque cualquier casa se queda pequeña para las cenas de nochevieja. Con los años juntarnos las tres generaciones se ha convertido en una misión realmente imposible.

Afortunadamente para los anfitriones, los primos nos hemos ido haciendo mayores y ahora no estamos toda la noche en la casa sino que hacemos otros planes tanto antes de la cena como después. El resultado es que cada uno llegamos y nos vamos a horas distintas. Es raro que coincidamos todos para la cena pero los abuelos están contentos porque esa noche todos pasamos al menos un rato con ellos. Y como no coincidimos todos a la vez, el número de puestos en la mesa (¡y de sillas!) que se necesitan no son tantos.

Entrada

La entrada está compuesta por un número indeterminado de casos de prueba, cada uno ocupando varias líneas.

La primera línea de cada caso de prueba contiene un único número N con la cantidad de comensales que pasaremos por la casa (hasta 100.000). A continuación aparecen N líneas con el instante de llegada y salida de cada uno (números distintos entre 0 y 10^9).

Tras el último caso de prueba viene una línea con un `0` que no debe procesarse.

Salida

Por cada caso de prueba se escribirá una línea con el número exacto de plazas que se necesitan para que los N comensales podamos permanecer sentados durante todo el tiempo que estamos allí, seguido de la cantidad de tiempo en la que la ocupación de la casa es máxima.

Ten en cuenta que en el mismo momento en el que una persona se va se puede sentar otra.

Entrada de ejemplo

```
2
0 10
5 15
5
0 20
1 3
15 18
3 5
18 19
0
```

Salida de ejemplo

```
2 5
2 8
```

J
Terminando los chalets

La urbanización de chalets está ya casi terminada. Solo queda que terminen sus tareas el electricista y el fontanero que, eso sí, se llevan bastante mal por lo que no pueden coincidir trabajando en el mismo chalet a la vez.

El tiempo que ambos necesitan en cada casa es el mismo pues depende exclusivamente de los metros construidos que tengan y da igual cuál de los dos haga la instalación antes, pues ninguno necesita que la parte del otro esté acabada para hacer la suya.

Teniendo en cuenta que el orden en el que se van terminando los chalets no es importante, debemos decidir el orden en el que cada uno hará su trabajo para que las obras de todos los chalets terminen cuanto antes.

Entrada

La entrada está compuesta por un número indeterminado de casos de prueba, cada uno ocupando dos líneas.

La primera línea de cada caso tiene el número N de chalets que hay que terminar (hasta 10^6). En la segunda línea aparecen N enteros con el tiempo que necesita trabajar tanto el fontanero como el electricista en cada chalet (números entre 0 y 10^{12}).

Salida

Por cada caso de prueba se escribirá la cantidad de tiempo necesario para terminar la construcción de todos los chalets.

Entrada de ejemplo

```
2
10 15
3
5 10 5
```

Salida de ejemplo

```
30
20
```

K
Adornos equilibrados

Después de varias horas dando vueltas por distintos puestos, acabas de hacerte con un flamante árbol de Navidad nuevo. Pero ahora se ha hecho tarde y tienes solo un momento para comprar los adornos que colgarás.

Estás enfrente de un puesto donde tienen un montón de ellos, todos en fila y todos al mismo precio. Con tu esquilmado presupuesto (no has escatimado en gastos con el árbol) sabes exactamente cuántos podrás adquirir y, para no perder mucho tiempo, has decidido que los cogerás todos contiguos en la fila de la tienda.

El problema es que también los colocarás en el árbol en el mismo orden y te da miedo que, al tener distintos pesos, el árbol se te desequilibre y se vuelque. Por eso, quieres comprar una secuencia de adornos tal que los primeros pesen exactamente lo mismo que los últimos.

Entrada

El programa deberá leer, de la entrada estándar, múltiples casos de prueba, cada uno ocupando dos líneas.

La primera contiene dos números. El primero, $2 \leq n \leq 700.000$, indica la cantidad de adornos para el árbol que tienen colocados, en fila, en la tienda. El segundo, $2 \leq c \leq n$, indica cuántos adornos vas a comprar.

La segunda línea del caso de prueba tiene n números, menores que 100, con el peso de cada uno de los adornos, en el orden en el que están colocados en la tienda.

Salida

Por cada caso de prueba el programa escribirá la posición del primer adorno que tienes que comprar, de modo que los c adornos a partir de ese puedan ser separados en dos partes consecutivas cuyos pesos sean exactamente el mismo. Si hay varias posibles respuestas, se dará aquella que tenga un mayor peso (¡burro grande, ande o no ande!). Si, aun así, sigue habiendo varias opciones se dará la menor posible (la situada antes en la entrada).

Se considera que el primer adorno de la lista está en la posición 1. Si no hay ninguna solución, se escribirá "`SIN ADORNOS`"

Entrada de ejemplo

```
4 2
1 2 2 1
4 2
1 2 1 2
7 3
2 1 1 6 3 3 6
```

Salida de ejemplo

```
2
SIN ADORNOS
4
```

L
25 años después

Hoy he descubierto que mis profesores son humanos. Te acostumbras a verles entrar en clase, soltar su rollete e irse y de alguna manera tu subconsciente piensa que en el momento que abandonan el aula su existencia se queda en suspenso hasta la siguiente clase en la que vuelven a salir de su hibernación para contarte la siguiente parte del tema. En definitiva, que son entes sin una vida más allá del aula. Sin pasado. Sin futuro.

Pero, como digo, hoy he descubierto que son humanos. Estábamos en un exámen y estaba mi profesor vigilando para que no copiáramos mientras charlaba animadamente con otro que había venido a hacerle compañía[1]. La conversación que estaban teniendo no tiene desperdicio, porque estaban hablando de sus "tiempos mozos" cuando ellos mismos hacían exámenes, así que no pude evitar prestar atención a lo que decían:

> — ...
>
> — Había un catedrático cuya única aportación a la asignatura era idear la pregunta del examen que no supiera responder nadie.
>
> — Igualito que ahora, que pasas por veinte versiones distintas de cada ejercicio hasta que das con uno que crees que van a saber hacer...
>
> — Ya te digo. Pues han pasado 25 años y todavía me acuerdo de la pregunta que, naturalmente, nadie supo responder: "¿Qué número divisible por 3 tiene un factorial que termina con 16 ceros?". Flipa.
>
> — ¡Ostras! ¡Qué bueno! Ese problema es chulísimo. De hecho está relacionado con esta asignatura. Igual alguno de estos sabría hacerlo —dijo señalándonos—. Ven que te lo explico.

Y se fueron hacia la pizarra; mientras uno explicaba, el otro se sorpendía de descubrir, 25 años después, cómo se resolvía el problema. Lo peor para mí fue que, sabiendo que la solución debería estar a mi alcance, me pasé el examen intentando resolverlo en lugar de centrarme en mis preguntas. Espero que no tengan que pasar 25 años para averiguar cómo se hace...

Entrada

La entrada estará compuesta de distintos casos de prueba, cada uno en una línea. Cada caso de prueba contiene dos números, d (entre 1 y 1000) y n (entre 0 y 10^{12}).

Tras el último caso de prueba viene una línea con dos ceros que no debe procesarse.

Salida

Por cada caso de prueba se escribirá una línea con el número $k > 0$ que, además de ser divisible por d, cumple que $k!$ termina con exactamente n ceros.

Si hay más de una solución posible se escribirá la más pequeña. Si no hay ninguna, se escribirá `NINGUNO`.

Entrada de ejemplo

```
4 0
5 0
5 2
3 16
0 0
```

[1] Basado en una historia real.

Salida de ejemplo

```
4
NINGUNO
10
72
```