Concurso de Programación
Ada Byron

Fase local - UCM

Cuadernillo de problemas

Facultad de Informática

UNIVERSIDAD COMPLUTENSE MADRID

16 de febrero de 2018

In almost every computation a great variety of arrangements for the succession of the processes is possible, and various considerations must influence the selections amongst them for the purposes of a calculating engine. One essential object is to choose that arrangement which shall tend to reduce to a minimum the time necessary for completing the calculation.
Ada Byron

16 de febrero de 2018

Listado de problemas

Autores de los problemas:

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

A
Semana de la Informática

En breve comenzará la *Semana de la Informática*. La facultad ha organizado multitud de actividades (conferencias, seminarios, talleres, jornadas de puertas abiertas). Yo he estado mirando la planificación y he ido apuntando todas las actividades que me parecen interesantes. El problema es que son tantas que algunas solapan en el tiempo, por lo que no podré asistir a todas. Lo que he pensado es convencer a algunos compañeros para que asistan a las actividades a las que no puedo ir yo para que entre todos asistamos a todas las actividades que me han parecido interesantes, y después compartamos lo aprendido.

Dada la lista de actividades, de las que conozco cuándo empiezan y terminan, ¿podrías ayudarme a calcular cuántos compañeros necesito como mínimo para cubrir todas las actividades? Puedes tener en cuenta que aunque ni mis compañeros ni yo tenemos el don de la ubicuidad, sí tenemos el poder de teletransportarnos, por lo que en cuanto termina una actividad podemos *saltar* a donde comience otra, sin perdernos nada.

Entrada

La entrada consta de una serie de casos de prueba. Cada uno comienza con una línea con el número N de actividades interesantes ($1 \leq N \leq 200.000$). A continuación aparecen N líneas, cada una con dos números que representan el momento de comienzo y de finalización de una actividad (el comienzo siempre es estrictamente menor que la finalización). Estos tiempos son números enteros entre 0 y 10^9.

La entrada terminará con un caso sin actividades ($N = 0$), que no debe procesarse.

Salida

Para cada caso de prueba se escribirá una línea con el mínimo número de compañeros que harán posible que, junto conmigo, podamos asistir a todas las actividades, y de tal forma que ninguno tenga que estar a la vez asistiendo a dos de ellas.

Entrada de ejemplo

```
3
1 5
3 10
6 12
2
5 10
1 5
3
1 5
2 6
3 7
0
```

Salida de ejemplo

```
1
0
2
```

B
Abuelas falsas

Hay una conjetura que dice que todas las abuelas del mundo cuando van a decir el nombre de un nieto, dicen primero el nombre de muchos otros nietos antes de decir el nombre correcto.

El gobierno ha ideado una prueba para saber si una persona mayor es abuela o no. La prueba consiste en enseñar una foto de un nieto y preguntar cómo se llama ese nieto, anotando los nombres que dice la mujer.

Si la mujer sólo ha dicho el nombre del nieto al final de la lista de nombres se considera que es una abuela verdadera, en caso contrario es una falsa abuela.

Entrada

La entrada comienza por el número N de pruebas realizadas a distintas personas mayores.

Cada una de esas pruebas ocupa una única línea. Comienza con el nombre real del nieto de la foto. A continuación viene un número positivo que indica la cantidad de nombres que dijo la abuela en cuestión (como mucho 100), al que le sigue cada uno de esos nombres.

Para evitar confusiones, tanto el nombre real como los nombres dichos por las abuelas aparecerán siempre en minúsculas y no contendrán espacios ni tildes o eñes. La longitud de cada nombre no excederá los 10 caracteres.

Salida

Por cada prueba se escribirá una única línea con la cadena `VERDADERA` si la persona encuestada es una abuela real y `FALSA` si es una farsante.

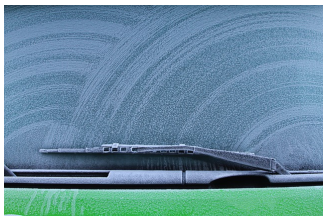
Entrada de ejemplo

```
3
mireia 5 ximo vicente maria vicente mireia
juan 2 juan maria
ximo 1 ximo
```

Salida de ejemplo

```
VERDADERA
FALSA
FALSA
```

C
Limpiaparabrisas de los híbridos



Me he comprado un coche híbrido. Es uno de esos que tiene un motor de combustión y un motor eléctrico, y dependiendo de las circunstancias usa uno u otro. La verdad es que son una chulada.

Pero cualquier cosa de mecánica que se tenga que hacer con él requiere un cuidado infinito. El otro día un camión me puso perdido el cristal delantero y al intentar limpiarlo vi que no tenía agua en el "*limpia*". Cuando llegué a casa, me acordé en el aparcamiento, y aprovechando que tenía en el maletero un par de garrafas grandes decidí llenarlo con el agua de un grifo que tenemos abajo. Al mirar las instrucciones del coche me asusté, porque decía que bajo ningún concepto derramara el agua fuera del depósito del "*limpia*", porque si caía en el motor eléctrico podía estropearlo (y, aún peor, sufrir una descarga). También decía que en el depósito entraban exactamente dos litros de agua. Yo tenía una garrafa de cinco litros y otra de cuatro, pero ninguna de dos, así es que preferí no arriesgarme y no llené el depósito.

Cuando lo conté en casa, se rieron de mí, porque me aseguraron que podría haber conseguido tener exactamente dos litros de agua en una de las garrafas. Les dije que las garrafas no tenían ningún tipo de escala, y yo no tenía nada con lo que marcar, pero lo único que conseguí fueron más risas. No entiendo nada.

Entrada

El programa tendrá que procesar múltiples casos de prueba. Cada uno comienza con un primer número l mayor que 0, indicando cuántos litros entran en el depósito del agua del coche, seguido del número $1 \leq r \leq 4$ de recipientes que tenemos a nuestra disposición.

En la siguiente línea aparecen r números indicando la capacidad, en litros, de cada recipiente disponible (entre 1 y 15).

Todos los números serán enteros. Además se garantiza que al menos uno de los recipientes tendrá una capacidad igual o mayor al del depósito del agua del coche.

La entrada termina con un `0`.

Salida

Para cada caso de prueba se escribirá "`SI`" si es posible conseguir tener exactamente l litros de agua en alguno de los recipientes y "`NO`" en otro caso. Ten en cuenta que no es posible marcar de ninguna forma los niveles de agua en los recipientes. Lo único que se puede hacer es vaciar o llenar un recipiente completamente, o hacer trasvases de uno a otro.

Entrada de ejemplo

```
2 2
5 4
4 2
12 15
0
```

Salida de ejemplo

```
SI
NO
```

D
El mejor de dos dados

Existe un juego para dos jugadores que se juega con dos dados. Cada jugador coge uno de ellos y lo lanza. El que obtiene la tirada más alta, gana esa ronda. Y, como es lógico, sale vencedor el que más rondas gana.

Desde un punto de vista de probabilidad, el juego no tiene ningún misterio si ambos jugadores utilizan dados idénticos. Ambos tienen las mismas probabilidades de ganar.

Lo divertido viene cuando los dados son distintos. En el momento de comenzar una partida, el jugador de más edad analiza ambos dados y decide con cuál jugará.

Entrada

La entrada está compuesta por varios casos de prueba, cada uno ocupando 3 líneas.

La primera línea contiene el número de caras de los dados ($2 \leq n \leq 100.000$). A continuación vendrán dos líneas con n números cada una que contienen los valores de cada cara (entre 1 y 10.000).

La salida termina con un caso de prueba con dados de 0 caras que no debe procesarse.

Salida

Por cada caso de prueba se escribirá `PRIMERO` si el jugador debe coger el primer dado para maximizar sus opciones de ganar, y `SEGUNDO` si es mejor el segundo dado.

Si no hay diferencia entre elegir uno u otro, se escribirá `NO HAY DIFERENCIA`.

Entrada de ejemplo

```
6
9 9 9 9 1 1
6 6 6 6 6 6
6
4 4 4 4 11 11
6 6 6 6 6 6
3
3 3 3
3 3 3
6
9 9 9 9 1 1
4 4 4 4 11 11
0
```

Salida de ejemplo

```
PRIMERO
SEGUNDO
NO HAY DIFERENCIA
SEGUNDO
```

E
Trim

Muchos lenguajes de programación proporcionan, a través de sus bibliotecas, la función `trim()`, que recibe una cadena de caracteres y devuelve otra tras eliminar, del inicio y del final, todos los espacios.

Algunos lenguajes disponen de una generalización de esa función, en la que el programador proporciona el carácter que se desea eliminar de los extremos, para que no sean obligatoriamente los espacios. Así, por ejemplo:

```
trim("lloron", 'l') = "oron"
 trim("oron", 'n') = "oro"
  trim("oro", 'o') = "r"
   trim("r", 'r') = ""
```

Entrada

El programa deberá procesar múltiples casos de prueba. Cada uno se compondrá de una única palabra (no vacía), de no más de 80 letras minúsculas del alfabeto inglés, escrita en una línea.

Salida

Por cada caso de prueba, el programa escribirá el *mínimo* número de llamadas a la función `trim()` generalizada que deberán realizarse de manera reiterada sobre la palabra del caso de prueba para que ésta sea consumida por completo. En cada invocación a `trim()`, el programa podrá escoger el carácter que considere más adecuado.

Entrada de ejemplo

```
lloron
baobab
caracola
aceptaelreto
```

Salida de ejemplo

```
4
4
6
10
```

F
La práctica de esteganografía

La *esteganografía* (del griego *steganos*, cubierto u oculto, y *graphos*, escritura) estudia técnicas que permitan ocultar mensajes dentro de otros (llamados portadores) de modo que no se perciba la presencia de los primeros. Es decir, procura ocultar mensajes dentro de otros de modo que el propio acto de la comunicación pase inadvertido para un probable intruso, que ni siquiera sabrá que se está transmitiendo información sensible.

En la escuela de esteganografía de Atenas, a Ocultaniakis le han puesto una práctica: tiene que encontrar la forma de esconder un número en un mensaje. Lo primero que se le ha ocurrido es introducirlo en una lista larga de números, pero le parece que esta solución es muy sencilla y recibirá poca nota en su evaluación. Así que ha decidido dar una vuelta de tuerca más a la esteganografía y hacer que el número oculto no esté presente en el mensaje, sino que haya que calcularlo buscando una clave oculta en el mismo.

La clave estará formada por una serie corta de números distintos. Estos aparecerán, posiblemente varias veces, dentro de una lista larga de números, el mensaje completo. El número secreto será la longitud de la subsecuencia más corta del mensaje que contenga los números de la clave en el mismo orden. En concreto, si la clave son los números c_1, c_2, ... , c_r, y el mensaje m_1, m_2, ... , m_n, una subsecuencia del mensaje contiene la clave si existen índices $1 \leq i_1 < \ldots < i_r \leq n$, tales que $m_{i_1} = c_1, \ldots, m_{i_r} = c_r$. Y en ese caso la longitud de la subsecuencia es $i_r - i_1 + 1$.

¿Sabrías recuperar la información escondida en un mensaje generado por Ocultaniakis?

Entrada

La entrada estará formada por una serie de casos de prueba. Cada caso está formado por 4 líneas: la primera contiene el tamaño R de la clave, entre 2 y 10 números; la segunda contiene los R números de la clave, todos distintos, en el orden en el que deben aparecer en el mensaje; la tercera contiene el tamaño N del mensaje, entre R y 250.000; y la cuarta contiene los N números del mensaje. Se garantiza que la clave siempre aparece al menos una vez oculta en el mensaje. Los números que aparecen tanto en la clave como en el mensaje están entre 1 y 10.000.

Salida

Para cada caso de prueba se escribirá una línea que contenga la longitud (número de elementos) de la subsecuencia más corta del mensaje que contenga la clave.

Entrada de ejemplo

```
3
1 2 3
10
5 1 3 2 1 7 3 2 3 8
2
3 1
4
31 3 5 1
```

Salida de ejemplo

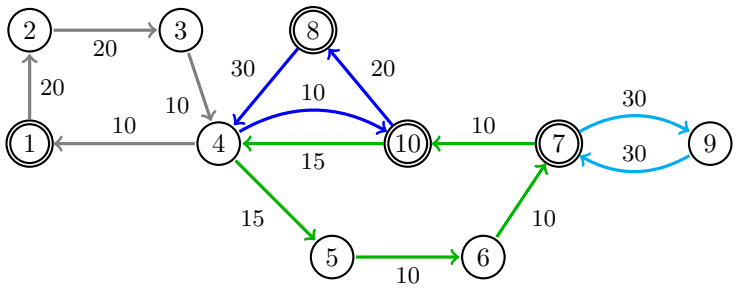
```
5
3
```

G
Huyendo de los zombis

Cuando termine el concurso tendrás que volver a casa en autobús. Como los organizadores son previsores, os han dado una planificación de los autobuses de la ciudad, donde se especifica para cada línea por qué paradas pasa y a qué hora. Madrid está sufriendo en la actualidad una plaga de zombis, por lo que, aunque las paradas de autobuses están rodeadas por una valla que supuestamente te protege de los zombis, tú prefieres pasar el menor tiempo posible esperando en una parada (aunque eso haga que el trayecto en autobús hasta tu casa sea más largo de lo necesario). Desafortunadamente, no hay ninguna línea de autobús que pase tanto por la parada de la facultad como por la parada enfrente de tu casa, por lo que por lo menos una vez tendrás que cambiar de autobús.

Ahora que estás algo ocioso mientras tus compañeros de equipo depuran la solución a un problema, quieres calcular la ruta que menor tiempo te hará esperar en las paradas de autobús.

En fin de semana las rutas de autobuses son siempre circulares y un autobús tarda exactamente una hora en completar la ruta, que repite de forma continua, empezando cada vuelta a las horas en punto. La siguiente figura muestra cuatro líneas de autobús. La primera, por ejemplo, pasa por las paradas con números 1, 2, 3 y 4. Saliendo en punto de la parada 1, pasa a los 20 minutos por la parada 2, a los 40 minutos por la parada 3 y a los 50 minutos por la parada 4 (la figura muestra lo que tarda un autobús en cubrir la distancia entre dos paradas consecutivas). Tras una hora vuelve a pasar por la parada número 1. La línea 2 comienza la ruta en la parada 10, la línea 3 comienza en la parada 8, y la línea 4 comienza en la parada número 7.



Con estas rutas, la mejor forma para ir de la parada número 1 (la facultad) a la parada número 10 (tu casa) consiste en tomar la línea 1 hasta la parada 4, esperar 25 minutos a que pase la línea 2 y llegar al destino (110 minutos después de salir). Si en la parada 4 quisiéramos tomar la línea 3, tendríamos que esperar 40 minutos (tardando en total 100 minutos). Para evitar el riesgo de que los zombis te devoren, es preferible la primera ruta.

Entrada

La entrada consiste en una serie de casos. Para cada caso, primero aparecen en una línea el número N de paradas en la ciudad (numeradas de 1 a N, con $2 \leq N \leq 1000$) y el número M de líneas de autobús ($2 \leq M \leq 100$). Después aparecen las descripciones de estas líneas, cada una ocupando una línea de texto. Para cada línea aparecen los números de las paradas por las que pasa (el primer número es la parada de comienzo, a las horas en punto) y entre cada par de números el tiempo que el autobús tarda de una parada a la siguiente. Recuerda que todas las líneas tardan una hora en dar la vuelta completa, por lo que el tiempo entre la última parada y la primera se puede calcular a partir de los demás.

Salida

Para cada caso de prueba se escribirá el tiempo mínimo de espera en las paradas de autobús para ir de la parada 1 (la facultad) a la parada N (tu casa). En estas paradas no hace falta esperar porque de la facultad puedes salir cuando pase el autobús que te interesa y al llegar a la parada N entras inmediatamente en tu casa. Puedes suponer que los autobuses cumplen el horario escrupulosamente, que tú tienes un reloj en hora, que los pasajeros suben y bajan de los autobuses instantáneamente (en

particular, si dos autobuses paran a la vez en la misma parada, puedes cambiar de uno a otro sin esperar) y que ir caminando entre dos paradas es tan peligroso que ni te lo planteas. Si es imposible llegar a casa desde la facultad, se escribirá `Hoy no vuelvo`.

Entrada de ejemplo

```
10 4
1 20 2 20 3 10 4
10 15 4 15 5 10 6 10 7
8 30 4 10 10
7 30 9
3 2
1 30 2
3 30 2
4 2
1 30 3
2 30 4
```

Salida de ejemplo

```
25
0
Hoy no vuelvo
```

H
Bingo infantil

Para que los niños que tiene en clase practiquen las restas que acaba de enseñarles, Mavi ha pensado en una versión especial del juego del Bingo. En la versión tradicional, cada jugador recibe un cartón con una serie de números, y se van extrayendo de un bombo bolas con números impresos hasta que alguien asegura que todos los números de su cartón han salido ya.

En la variante que ha pensado Mavi, en cada jugada extraerá dos números en lugar de solo uno. El valor jugado, que los niños tendrán que tachar de sus cartones, es la resta del mayor menos el menor. Tras cada jugada, los dos números serán incorporados de nuevo al bombo, en contra de lo que ocurre en el juego tradicional.

Aunque la idea es interesante, Mavi se enfrenta a un problema. El bingo que va a utilizar lleva en el armario de la clase muchos años y ha pasado por muchas manos... algunas un poco descuidadas que han hecho que se pierdan bolas. De modo que necesita saber la lista de números que pueden "salir" en su particular bingo, para ponerlos en los cartones y que todos tengan la posibilidad de ganar.

Mavi es consciente de que seguramente algunos números tengan más posibilidades de salir que otros, pero no le importa mucho. De hecho más bien lo considera una virtud, porque así podrá crear cartones con números más probables para los niños que restan con dificultad y que tengan también posibilidades de ganar.

Entrada

Cada caso de prueba comienza con un número indicando cuántas bolas quedan aún en el bingo de la clase (al menos 2). A continuación aparece el número de cada una de ellas. Todos los números son valores entre 1 y 2.000 y no hay ninguno repetido.

La entrada termina con un 0 que no debe procesarse.

Salida

Para cada caso de prueba, el programa deberá escribir, por la salida estándar los números que pueden formarse, ordenados de menor a mayor y separados por un espacio. No debe haber espacio tras el último número.

Entrada de ejemplo

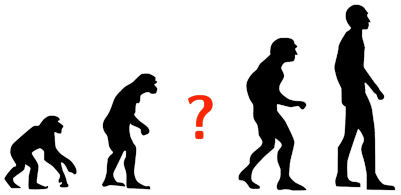
```
4
1 3 4 5
3
4 1 8
0
```

Salida de ejemplo

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

I
Eslabones perdidos

Según la teoría de la evolución, todas las especies de seres vivos han surgido como resultado de las mutaciones progresivas, a lo largo de millones de años, de las primeras bacterias. Se forma así un "árbol de especies" con el que se podría seguir la pista de todos los pasos intermedios por los que una especie ha pasado hasta llegar a ser como es.



Los registros fósiles dan muestra de muchas de esas especies. Sin embargo, hay muchas *formas transicionales* que no se han encontrado, y que a menudo son llamadas *eslabones perdidos*.

Entrada

El programa deberá procesar múltiples casos de prueba recibidos por la entrada estándar. Un caso de prueba está compuesto de una primera línea indicando el número e de especies conocidas (entre 1 y 10.000) y el número r de relaciones (diferentes) entre ellas.

A continuación vendrán r líneas, cada una con dos números distintos $1 \leq o,d \leq e$ indicando que la especie número o es considerada la predecesora directa de la especie d.

Se garantiza que cada especie tendrá, como mucho, una especie que la precede, que no se formarán ciclos y que cada relación aparecerá en la entrada una única vez.

La entrada termina con un `0`.

Salida

Para cada caso de prueba, el programa escribirá "`TODAS`" si se conocen todas las especies de modo que es posible seguir el rastro desde cada una hasta la primera bacteria y "`FALTA ALGUNA`" en otro caso.

Entrada de ejemplo

```
3 1
1 2
3 2
1 2
1 3
0
```

Salida de ejemplo

```
FALTA ALGUNA
TODAS
```