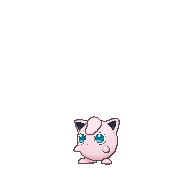
Compilación de problemas de concurso

Estrategias Algorítmicas, primavera 2022, ITESO





Problemas elaborados por Iván Piza y Luis Gatica

Documento elaborado por Luis Gatica

**Lineamientos**

1. Las soluciones de cada problema deberán codificarse en Java, C o C++.
2. A cada problema resuelto le corresponde un solo archivo que contiene el código fuente que resuelve el problema.
3. En el caso de Java, el nombre de la clase principal que le corresponde a la solución de cada problema siempre será *Main*.
4. Utilice la entrada y la salida estándar (*Scanner*, *BufferedReader*, *System.out*…) y NUNCA utilice funciones de manejo de archivos (su solución será descartada automáticamente, y en realidad ni siquiera es necesario el manejo de archivos en los concursos de programación).
5. No defina paquetes o *namespaces*.
6. Si el problema exige efectuar un ordenamiento, tiene permitido apoyarse de los métodos disponibles en la API de Java (o la *STL* en C++), o implementar su propio método.
7. El puntaje asignado a cada problema estará en función del porcentaje de casos de prueba correctos.
8. Si el tiempo de ejecución de una solución rebasa el tiempo previsto para ella (definido por el profesor), la solución será rechazada por el juez.
9. Puede hacer pruebas localmente compilando su código en línea de comandos con javac Main.java y ejecutándolo con la línea java Main < input.txt > output.txt, donde *input.txt* es un archivo de texto ya existente con la entrada deseada (que será tomado en la ejecución como la entrada estándar *System.in*) y *output.txt* será un archivo creado en la ejecución de Main que contendrá la salida estándar *System.out*.

**Índice**

[ADN 4](#_Toc79520443)

[Abrazos correspondidos 6](#_Toc79520444)

[Asesorías personalizadas 7](#_Toc79520445)

[Bolitas de amigas 8](#_Toc79520446)

[Cartas de San Valentín 10](#_Toc79520447)

[Dé de dinero 11](#_Toc79520448)

[Diámetro de Redes Sociales 12](#_Toc79520449)

[Estaciones de ser vicio 13](#_Toc79520450)

[Estantería de Babel 15](#_Toc79520451)

[Formación por estaturas 16](#_Toc79520452)

[Gastos perplejos 17](#_Toc79520453)

[Justicia laboral 18](#_Toc79520454)

[Koopeedo 20](#_Toc79520455)

[La florería 21](#_Toc79520456)

[Lista telefónica 22](#_Toc79520457)

[Los caprichos del chef 23](#_Toc79520458)

[Minúsculas 24](#_Toc79520459)

[Multinivel 25](#_Toc79520460)

[Noticias falsas 27](#_Toc79520461)

[Permutación común 29](#_Toc79520462)

[Productos más vendidos 30](#_Toc79520463)

[Que no se te vaya el tren 31](#_Toc79520464)

[Ruta crítica 32](#_Toc79520465)

[Secuencias 33](#_Toc79520466)

[Subsecuencias distintas 35](#_Toc79520467)

[Sucursales convenientes 36](#_Toc79520468)

[Utilidades 37](#_Toc79520469)

[Un tercer cuerpo 39](#_Toc79520470)

# ADN

Este problema consiste en implementar una estructura jerárquica para almacenar una lista de secuencias ADN, semejante a la que utilizan los motores de búsqueda y diccionarios electrónicos. Una aplicación de esta estructura es poder filtrar sólo los elementos de la lista que comiencen con los caracteres que el usuario haya escrito hasta el momento.

Para entender mejor, analice la siguiente estructura que se construyó a partir de la lista de secuencias: ADGT, AADG, GTDD, TTAD, GAGT, GDAT, AAAD.

TTAD

GTDD

ADGT

GAGT

GDAT

AADG

AAAD

Como se puede apreciar, cada nodo interno de la estructura puede tener hasta 4 hijos; cada uno corresponde a las letras A, D, G y T. Para encontrar la secuencia AAAD en la estructura, necesitamos realizar sólo 3 comparaciones: con ADGT, AADG y AAAD. Si deseamos buscar la secuencia GDAA, con sólo 2 comparaciones nos damos cuenta que no existe: con GTDD y GDAT.

**Input**

La primera línea contiene un número entero C > 0 que denota el número de casos de prueba. Cada caso de prueba comienza con una línea que contiene un número entero N (2 ≤ N ≥ 100) que denota el número de secuencias ADN a introducir. Le siguen N líneas, cada una contiene una secuencia ADN única para el caso de prueba, formada por las letras mayúsculas A, D, G, T. Después, una línea con un número entero B que denota el número de búsquedas a realizar. Le siguen B líneas, cada una contiene una secuencia ADN que puede encontrarse o no en la lista anterior. Todas las secuencias incluidas en el mismo caso de prueba tienen la misma longitud.

**Output**

Para cada caso de prueba imprime una línea formada por una lista de B de números enteros separados por comas que denotan el número de comparaciones entre secuencias ADN que se tuvieron que efectuar para realizar cada búsqueda a través de la estructura jerárquica.

**Sample Input**

1

7

ADGT

AADG

GTDD

TTAD

GAGT

GDAT

AAAD

4

TTAD

AAAD

GAGT

GDAA

**Sample Output**

1 3 2 2

# Abrazos correspondidos



¿A quién no le gusta un buen abrazo? O un abrazo dentro de un abrazo dentro de un abrazo. En este problema se trata de saber si todos los abrazos en una cadena son correspondidos. Un abrazo es enviado con el código > mientras que es devuelto con el código <. Un abrazo inmediatamente correspondido se lee así: ><. Pero un abrazo puede ser aplazado si hay otro abrazo “anidado” y correspondido: >><<. Inclusive un abrazo correspondido puede ser seguido de otros abrazos correspondidos: >><<><. En cambio, hay un problema cuando un abrazo queda sin corresponder: >>><< o si se “corresponde” un abrazo que no fue dado: <><. Determine si todos los abrazos fueron correspondidos o no.

**Input**

Este problema tiene un número no explicitado de casos de prueba, cada uno en una línea terminada con un salto de línea. Cada caso de prueba consta de una secuencia (de longitud no explicitada) de los caracteres > y <. Puede asumir que cada caso contiene al menos un carácter antes del salto de línea.

**Output**

Por cada caso de prueba imprima una sola línea (terminada con salto de línea). Esta línea incluirá -1 si todos los abrazos fueron correspondidos. Si no lo son, indique dónde en la cadena de abrazos se encuentra la acción problemática (último abrazo enviado y no correspondido o primer abrazo “correspondido” pero no enviado). Los índices de la cadena están basados en 0.

|  |  |
| --- | --- |
| **Sample Input**  ><><><  ><<  ><>  >><<  >>><  >>><<<>  ><>><<>>><<<>>><><<< | **Sample Output**  -1  2  2  -1  1  6  -1 |

# Asesorías personalizadas

El profesor titular de la materia Cálculo Renal Avanzado se dio cuenta que muchos de sus alumnos están teniendo problemas para comprender la mayoría de los temas y sus promedios actuales son muy bajo; por otro lado, existe otro grupo de alumnos que se está aburriendo porque considera que los temas vistos han sido muy sencillos o el ritmo muy lento; el promedio de estos alumnos es alto. Por lo tanto, el profesor decidió contratar a dos asesores externos. Uno de ellos, experto en pedagogía, asesorará con mucha paciencia a la mitad de alumnos con promedio más bajo. El otro, experto en la materia, proporcionará problemas más retadores a la mitad de alumnos con promedio más alto. Como los grupos siempre tienen un número impar de alumnos, quedará un alumno volando. Ese será asesorado por el profesor titular de manera individual. Tu trabajo consiste en apoyar al profesor titular en encontrar a dicho alumno en el menor número de selecciones fallidas.

Para comprender la metodología a seguir, considera el siguiente ejemplo. Hay cinco alumnos y sus calificaciones actuales son: 60, 70, 80, 90, 100. Seleccionas el de la mitad (80) y diste con el alumno. El problema es que las calificaciones no siempre están ordenadas, ya que el ordenamiento original es por nombre del alumno. Considera el siguiente ejemplo que tiene tres selecciones fallidas:

1. Lista original: 100, 80, 60, 90, 70
2. Seleccionas el de la mitad 60: fallida (te quedaste corto).
3. Filtras la lista: 100, 80, 90, 70.
4. Seleccionas el de la mitad 90: fallida (te pasaste).
5. Filtras la lista: 80, 70.
6. Seleccionas el de la mitad 70: fallida (te quedaste corto).
7. Filtras la lista: 80
8. Seleccionas el de la mitad 80: lo encontraste.

**Input**

La primera línea contiene un número entero T > 0 que denota el número de casos de prueba. Cada una de las Tlíneas siguientes comienza con un número entero impar N, tal que 3 ≤ N < 100, seguido por N números enteros diferentes en el rango [0...100].

**Output**

La salida consiste en T líneas que contienen cada una dos números enteros C, F que denotan, respectivamente, la calificación del alumno a atender por el profesor titular y el mínimo número de intentos fallidos para dar con él.

**Sample Input**

2

5 60 70 80 90 100

5 100 80 60 90 70

**Sample Output**

80 0

80 3

# Bolitas de amigas

En cualquier primaria las niñas suelen formar grupos de amigas que llaman “bolitas”. En algún planeta extraño, hay tan poco oxígeno que no puede haber más de dos niñas juntas al mismo tiempo (así que eso de *bolita* queda más como una metáfora o una añoranza de otro espacio u otro tiempo). Dos niñas pueden hablar entre sí o no: inclusive, una puede hablarle a la otra sin que la otra le hable. Para este problema consideraremos que dos niñas *a* y *b* tienen una relación *aRb* si y sólo si existe una cadena de niños hablantes que empieza con *a* y termina con *b* (siendo *a* la primera en hablar y *b* la última en escuchar; cada niña “intermedia” escucha primero y habla inmediatamente después; puede que no haya niñas “intermedias” si *a* habla directamente con *b*). Nótese que puede haber relación *aRb* sin que exista relación *bRa*. Admitiremos, para todo caso, una cadena vacía de hablantes, lo cual implica que para toda niña *n* se cumple la relación *nRn*. En ese planeta extraño, se considera una bolita de amigas a cada conjunto de niñes *C* (con tamaño máximo) tal que para cualquier par de niñas *u*, *v* en *C*, existen las relaciones *uRv* y *vRu*. Note que esto, junto a la restricción de “maximalidad”, implica que una niña pertenece sólo a una bolita.

Por ejemplo: si Ana le habla a Bety, Bety le habla a Caro, Caro le habla a Ana, Danny le habla a Eliza, Eliza le habla a Danny y nadie habla con Fer ni Fer con nadie, aunque Ana le hable a Danny las *bolitas* son tres: (1) Ana, Bety, Caro; (2) Danny, Eliza; (3) Fer.

A partir de los nombres de las niñas en una primaria de ese planeta y de una lista de pares de niñas (en que la primera le habla a la segunda), determine cuáles son las *bolitas* de amigas.

**Input**

La primera línea contiene un número entero T > 0 que denota el número de casos de prueba. Cada uno de los Tcasos está compuesto por: una línea con un entero N tal que 1 < N <= 1000 que indica cuántas niñas hay en total, seguida de una línea con un entero H tal que 1 < H <= N2 - N que indica cuántos pares de niñas hablan directamente entre sí cuando están juntas, seguida de N líneas cada una con el nombre de una niña. Cada caso termina con H líneas donde cada una incluye los nombres de dos niñas que hablan directamente: primero el nombre de la niña que habla y luego el nombre de la niña que escucha, separados por un espacio.

Los nombres de las niñas de un solo caso nunca se repiten. Cada nombre está compuesto de como máximo 15 letras del alfabeto inglés, todas mayúsculas (como si los hubieran escrito señoras de Piolín gritando en facebook).

**Output**

La salida de cada caso estará compuesta por varias líneas: cada línea incluirá la lista de nombres de las niñas en una bolita. Si una bolita contiene más de una niña, los nombres de esa bolita se ordenarán alfabéticamente de menor a mayor. Si en un caso hay más de una bolita, las bolitas se ordenarán alfabéticamente de menor a mayor, considerando como único criterio el nombre de la primera niña en esa bolita. Cada par de casos estarán separados entre sí por una línea en blanco.

**Sample Input**

2

6

6

DANNY

ANA

CARO

FER

BETY

ELIZA

ANA BETY

BETY CARO

CARO ANA

ANA DANNY

DANNY ELIZA

ELIZA DANNY

3

2

CGIRL

AGIRL

BGIRL

CGIRL BGIRL

BGIRL CGIRL

**Sample Output**

ANA BETY CARO

DANNY ELIZA

FER

AGIRL

BGIRL CGIRL

# Cartas de San Valentín

En una universidad no muy lejana, la gente se envía cartas el día de San Valentín. Usted recibe una lista de enteros que indica, cada uno, que una carta fue enviada a una persona. Cada persona es identificada por un solo entero, y puede haber personas que no reciban ninguna carta. Usted debe identificar cuál es la persona que recibirá más cartas. Como el *capitalismo heteropatriarcal tortuga ninja adolescente mutante* promueve el espectáculo y la exaltación de algunos personajes, puede estar seguro de que siempre habrá una persona que reciba más cartas que cada una de las otras personas. (¿Víctimas del patriarcado?)

**Input**

Este problema tiene un número no explicitado de casos de prueba. Cada caso de prueba está compuesto por una línea que incluye un número no explicitado de enteros no negativos (al menos 10) separados entre sí por un solo espacio.

**Output**

Para cada caso de prueba, imprima el entero correspondiente a la persona que recibirá más cartas en San Valentín, seguido de un salto de línea.

|  |  |
| --- | --- |
| **Sample Input**  5 2 4 4 1 3 5 8 10 5 1 6 7  1 2 3 4 5 6 7 8 9 10 11 10  6 1 6 1 6 1 6 1 6 1 61 1 | **Sample Output**  5  10  1 |

# Dé de dinero

C-lena es un robot de un futuro muy próximo que no tiene una inteligencia artificial muy… inteligente. C-lena debe calcular, a partir de una cantidad infinita de monedas de ciertas denominaciones, cómo devolver el cambio D utilizando el total de monedas más pequeño posible. ¡Ayude a C-lena a calcular la cantidad de monedas a devolver!

**Input**

La primera línea de la entrada contiene un número entero T que representa el número de casos de prueba. Cada caso comienza con una línea que contiene un número entero N (2 <= N <= 15) que representa el número de denominaciones, seguido de otra línea que contiene el número D. Después, N líneas, donde la línea i incluye un solo entero que representa el valor de una sola moneda de la denominación i.

**Output**

Para cada caso de prueba, imprima un entero que indique la mínima cantidad de monedas de las denominaciones correspondientes necesaria para devolver el cambio D. Si no es posible construir el cambio D con monedas de las denominaciones disponibles, imprima el entero -1.

**Sample input**

2

3

11

1

3

5

2

7

4

10

**Sample output**

3

-1

# Diámetro de Redes Sociales

Encontrar el diámetro de una red social que sigue el modelo *Enforced-Mutual-Friend*, como Facebook, nos permite saber el número de veces que una publicación dada tiene que ser compartida para que cada miembro de la red pueda enterarse de ella. Como un ejemplo, considere la red social ilustrada en la siguiente figura:

3

5

1

4

2

6

El diámetro de esta red social es cuatro, que corresponde a la distancia más corta entre 2 y 6. Esto significa que, si 2 pretende compartir a todos (*broadcast*) un mensaje, el mensaje tiene que ser compartido al menos tres veces: por 4, 3 y 5, para que todos en la red puedan leerlo. El problema consiste en encontrar el diámetro de una red social dada. Asuma que cada miembro de la red social está conectado con al menos otro miembro distinto de sí mismo.

**Input**

La primera línea contiene un número entero *N* > 0 que denota el número de casos de prueba. Cada una de las siguientes *N* líneas empieza con dos enteros *F*, *P*, 1 ≤ *F* ≤ 20, denotando respectivamente el tamaño de la red social y el número de amistades o conexiones entre pares de personas; luego, P líneas que contienen cada una dos enteros *a*, *b*, 1 ≤ *a*, *b* ≤ *P*, indicando que la persona cono indexada como *a* es amiga de la persona indexada como *b*, tal que *a* ≠ *b*.

**Output**

La salida consiste en *N* líneas, cada una conteniendo el diámetro D > 0 del caso de prueba correspondiente.

**Sample Input**

1

6 6

1 2

1 3

3 4

2 4

3 5

5 6

**Sample Output**

4

# Estaciones de ser vicio[[1]](#footnote-1)

A un par de amigos les gusta hacer un recorrido arrabalero de bar en bar, pero sólo pueden ir de un bar a otro si tienen conexión directa entre sí. En tal caso, la conexión es mutua. El costo de recorrer cualquier conexión directa entre bares es C >= 1. En caso de tener que decidir a cuál bar visitar primero, los amigos escogen siempre el bar con el nombre alfabéticamente menor.

Dada una lista de bares, una lista de las conexiones directas entre bares y un bar inicial B, determine la distancia mínima para llegar de B a todo otro bar.

**Input**

La primera línea contiene un número entero T > 0 que denota el número de casos de prueba. Cada caso empieza con una línea que contiene un entero N > 1, el número de bares, seguido de una línea con un entero 1 <= M <= (N2 – N) / 2, seguido de una línea con el entero C >= 1. Después, N líneas cada una con el nombre de un bar (máximo 15 mayúsculas), seguidas de M líneas cada una con el nombre de dos bares con conexión directa entre sí, separados por un espacio. Cada caso termina con una línea que contiene el nombre del bar B.

**Output**

Por cada caso de prueba, imprima una sola línea, compuesta por la distancia mínima de B a cada bar (considerados en orden alfabético) incluyéndose a sí mismo (en cuyo caso la distancia es 0). Si un bar es inalcanzable desde el bar B, se tomará -1 como su distancia para marcar que es inalcanzable. Separe cada par de distancias con un espacio.

**Sample Input**

2

7

7

30

A

B

C

D

E

F

G

A B

A F

B D

D E

D F

E F

C F

A

3

1

7

AAR

BAR

CAR

AAR BAR

BAR

**Sample Output**

0 30 60 60 60 30 -1

7 0 -1

# Estantería de Babel

En la literatura producida por Jorge Luis Borges, *El libro de arena* es un libro de contenido infinito. Además, *La biblioteca de Babel* es una biblioteca de tamaño también infinito. Un librero con mucho dinero decide crear una *estantería de Babel*, que tiene infinitos niveles. El primer nivel tiene un solo libro y cada nivel puede contener a lo mucho el doble de libros que el nivel de arriba, y el estante puede crecer indefinidamente hacia abajo y hacia los lados (el dinero resuelve muchas cosas). El librero es muy quisquilloso y decide acomodar los libros de una forma muy particular. El primer libro se coloca hasta arriba del primer nivel. Cada libro siguiente que se añade al estante debe quedar en el nivel inmediatamente debajo de un libro ya colocado, a su izquierda o a su derecha. Ningún “lugar” puede ser ocupado por más de un libro al mismo tiempo. El libro designa que los libros con títulos “menores” van a la izquierda y los libros con títulos “mayores” van a la derecha, siguiendo un orden lexicográfico (el orden que cierto lenguaje de programación de nombre Java utilizaría para ordenar una lista de datos de tipo *String*).

Tu deber es recibir la lista de títulos de los libros en el orden en que serán colocados en el estante para, después, mostrar cómo quedaron acomodados. El orden de tal despliegue será mostrar los libros de cada nivel desde el último (más abajo) hasta el primero (más arriba), y cada nivel se recorrerá desde la derecha hacia la izquierda.

**Input**

La primera línea contiene un entero N que denota el número de libros. Cada una de las N líneas siguientes consiste en una cadena de texto (el título del libro respectivo). Todas las líneas están terminadas por un salto de línea. Los títulos de los libros sólo incluyen mayúsculas y minúsculas del alfabeto inglés, números y espacios.

**Output**

La salida consiste en N líneas. Cada línea contiene el título de un libro, en el orden de despliegue que el librero ha decidido. Cada línea está terminada por un salto de línea, incluyendo la última.

**Sample Input**

10

La invenciOn de la soledad

ExtraNando a Kissinger

Los versos satAnicos

La vida instrucciones de uso

Ulises

Madame Bovary

Historias de cronopios y famas

En busca del tiempo perdido 1

El libro de arena

Pedro PAramo

**Sample Output**

Pedro PAramo

Madame Bovary

El libro de arena

Ulises

La vida instrucciones de uso

Historias de cronopios y famas

En busca del tiempo perdido 1

Los versos satAnicos

ExtraNando a Kissinger

La invenciOn de la soledad

# Formación por estaturas

Estás a cargo de un grupo de alumnos de primaria quienes se tienen que formar por estatura cuando el director toca la campana para salir del salón. Los alumnos salen formados en una fila siguiendo un orden arbitrario; tal vez los que estaban más cerca de la puerta están hasta adelante. Como conoces las estaturas de tus alumnos, sabes cuál debe ser el orden definitivo. Sin embargo, no quieres crear un caos afuera de tu salón. Para ordenar, quitas de su lugar actual a un alumno a la vez y lo llevas hasta delante hasta encontrar a uno más chaparrito. Todos los alumnos más altos que estaban adelante se mueven un paso atrás. Este proceso lo comienzas con el segundo alumno de la fila original, y de ahí hacia atrás. Cada alumno está identificado por un número de expediente único.

**Input**

Cada caso de prueba está descrito en 3 renglones. El primer renglón contiene un número entero N, tal que 1 < N < 10000, que denota el número de alumnos de la fila. Le sigue una lista de N números de expediente separados por espacios; cada expediente es un número entero en el rango [0...9999]. Esta lista representa el acomodo de los alumnos al salir del salón. El tercer renglón contiene otra lista con el mismo formato, que denota el acomodo esperado de los alumnos: por estaturas. El carácter 0 representa el fin de la entrada y no se debe procesar.

**Output**

Por cada caso de prueba, imprime una lista separada por espacios con los expedientes de los alumnos que tuvieron que salirse de su lugar para llegar al acomodo esperado. Si nadie se tuvo que salir de su lugar, imprime “Ordenado”.

**Sample Input**

3

1 0 2

0 1 2

6

200 300 100 400 600 500

100 200 300 400 500 600

4

10 12 8 15

10 12 8 15

0

**Sample Output**

0

100 500

Ordenado

# Gastos perplejos

En estos tiempos de aumento de precios en productos y servicios básicos, es difícil mostrarse indiferente cuando uno es informado de ciertos gastos (de cuyo adjetivo no quiero acordarme) tan elevados e innecesarios que se llevan a cabo en diferentes dependencias de gobierno.

Dicen las malas lenguas que cada dependencia tiene la misión de gastar la mayor parte del recurso que le ha sido entregado para no reducir la cantidad en la próxima asignación. En este problema, usted es el director de finanzas de una dependencia. Se le ha dado la misión de ejercer todo el dinero que sobró, sin endeudarse y eligiendo productos y servicios de un catálogo. Si logra identificar un subconjunto de productos y/o servicios cuyo costo total sea igual al dinero a gastar, usted será recomendado para ser el próximo Secretario de Finanzas del Estado.

**Input**

Cada línea denota un caso de prueba y contiene una lista de números enteros positivos separados por espacios. El primer número es el dinero a ejercer. El segundo número es la cantidad de productos y servicios disponibles. Los siguientes números representan los costos de los productos y servicios. La última línea contiene 0 y no será procesada. La cantidad a ejercer nunca será mayor que un millón de pesos. El número de productos y servicios disponibles no será mayor que 100 y no costará más de cien mil pesos cada uno.

**Output**

Por cada caso de prueba, deberá desplegar el mensaje **Secretario de Finanzas**, si es posible seleccionar un conjunto de productos y servicios cuyos costos suman la cantidad a ejercer. Si no lo es, deberá desplegar el mensaje **Ni modo**.

**Sample Input**

10 5 3 9 6 2 4

500 4 250 200 150 400

0

**Sample Output**

Secretario de Finanzas

Ni modo

# Justicia laboral

En cierta organización se considera una injusticia que un empleado perciba más que su supervisor inmediato. Tu trabajo es determinar si existe justicia o no, dados los sueldos de cada empleado y la información del número de empleados a cargo de cada supervisor. Si no existe justicia, deberás intercambiar los sueldos de cada supervisor con su subordinado mejor pagado si sobrepasa su sueldo. La resolución de injusticia deberá comenzar con el *sub-organigrama* que dirige el último supervisor. Luego con el sub-organigrama dirigido por el supervisor anterior, y así, sucesivamente hasta llegar al organigrama total, encabezado por el director general.

**Input**

La primera línea contiene un número entero T > 0 que denota el número de casos de prueba. Cada caso de prueba comienza con un número entero N ≥ 2 que denota el número de empleados de una organización, seguido por N líneas que contiene cada una un par de números enteros S, C, tal que S denota el sueldo del empleado, y C el número de empleados a su cargo. La información de los empleados a su cargo se especifica en los renglones inmediatamente siguientes, antes que la del resto de los empleados de su mismo nivel.Para mayor entendimiento, el segundo caso de prueba se puede representar con la figura siguiente.

130

140

80

120

155

146

El proceso comienza con el *sub-organigrama* dirigido por el supervisor 140. Ambos subordinados perciben más que él. Por tanto, intercambia su sueldo con el mejor pagado (155). El siguiente *sub-organigrama* a revisar es el encabezado por el director general (130). Para este caso, intercambia su sueldo con el supervisor anterior (155). Pero ahora, el supervisor anterior vuelve a ganar menos (130) que sus dos subordinados (140, 146), por tanto, se intercambia con 146, quedando así:

155

146

80

120

140

130

**Output**

Para cada caso de prueba imprime JUSTICIA si no existe ningún empleado que gane más que su supervisor; en otro caso, imprime N líneas de números enteros S, tal que S denota el sueldo de cada empleado después de impartir justicia. El orden de impresión de empleados es el mismo que se siguió en la entrada. Separa cada caso de prueba con una línea en blanco.

**Sample Input**

2

3

1000 2

850 0

950 0

6

130 3

120 0

140 2

155 0

146 0

80 0

**Sample Output**

JUSTICIA

155

120

146

140

130

80

# Koopeedo

Koopeedo tiene como misión emparejar a todos los habitantes del planeta, sabiendo que existe un número par de ellos. El único criterio que utilizará para determinar si dos personas pueden ser pareja es qué tan cerca viven uno del otro, en términos de zonas horarias. Una zona horaria se indica como un número entero en el conjunto {-11, -10, …, 0, …, 11, 12}. La cercanía entre dos personas que viven en zonas horarias *j*, *k* es el mínimo entre |*j* – *k­*| y 24 – |*j* – *k*|. Tienes que ayudar a Koopeedo a encontrar una partición de N personas en N/2 parejas de forma que se minimice la cercanía total.

**Input**

La primera línea de la entrada contiene un número entero T que representa el número de casos de prueba. Cada una de las siguientes T líneas comienza con un número entero par N (2 ≤ N ≤ 106) que representa el número de personas a emparejar, seguido por N enteros separados con espacios, cada uno entre -11 y 12 y representando una zona horaria.

**Output**

Para cada caso de prueba debes imprimir un entero que representa la mínima cercanía de una partición en parejas del conjunto de entrada.

**Sample input**

3

6 -3 -10 -5 11 4 4

2 -6 6

8 0 0 2 0 0 0 0 1

**Sample output**

5

12

1

# La florería

Todo el mundo se vuelve loco en San Valentín, incluidos quienes redactan problemas de concurso, o quienes visitan o atienden una florería llena de gente. Hay una en especial que da un papel con un número *X* a cada cliente que entra. Los clientes están formados en una fila ordenada por orden de llegada. Pero cada vez que va a atender a alguien, el florista pide que la fila se reacomode. La persona aún no atendida cuyo número *X* sea el menor, irá avanzando en la fila desplazando uno por uno a quienes están delante de él, hasta llegar a la primera posición de la fila, donde será atendido por el florista. Si hay más de una persona con el mismo número *X*, tiene prioridad la que ya está más cerca del florista (que está delante de la primera posición de la fila) y las otras personas con ese mismo número NO generarán desplazamientos.

Note que cada vez que un cliente desplaza a los otros también se está desplazando él, y que aunque su desplazamiento sea más largo cuenta como uno solo. Lo comento porque como usted no tiene a quién llevarle flores, se dedica a contar cuántos desplazamientos suceden.

**Input**

Este problema tiene un número no explicitado de casos de prueba. Cada caso está compuesto como sigue. La primera línea contiene un número entero *N*>5 que denota el número de clientes. Las siguientes *N* líneas contendrán cada una el número entero *X* del cliente respectivo en orden de llegada (primero el que llegó primero). Asuma tranquilamente que 0<=*X*<=*N* para todo *X*.

**Output**

La salida de cada caso estará compuesta por una sola línea (terminada siempre en salto de línea) con el formato siguiente “*Meterse en la fila provoca* D *movimientos.*” (sin incluir las comillas) donde *D* es el número de desplazamientos realizados.

|  |  |
| --- | --- |
| **Sample Input**  5  3  4  2  1  5  6  0  1  2  3  4  6  5  5  4  3  2  1 | **Sample Output**  Meterse en la fila provoca 7 movimientos.  Meterse en la fila provoca 0 movimientos.  Meterse en la fila provoca 14 movimientos. |

# Lista telefónica

Debido al gran número de reclamaciones, cierta compañía telefónica está siendo obligada a invertir más en la mejora de sus servicios. Para eso, decidió disminuir el presupuesto de algunos departamentos para aumentar el de otros más importantes. Uno de los departamentos que tendrá su presupuesto reducido es el de impresión de listas telefónicas.

Con un presupuesto reducido, el departamento de impresión de listas telefónicas no puede comprar tóner suficiente para imprimir las listas completas. Como los números telefónicos se imprimen alineados en la vertical, se sugirió la siguiente solución: a partir del segundo número impreso, los dígitos iniciales del próximo número a ser impreso deben coincidir con los números de arriba que son omitidos, quedando solo un espacio en blanco. Por ejemplo, para los números 535456, 535488, 536566 y 835456 la impresión es la siguiente:

5 3 5 4 5 6

8 8

6 5 6 6

8 3 5 4 5 6

Obsérvese que esta impresión economizó la impresión de 6 caracteres. Como nuevo empleado de la compañía, usted deberá construir un programa que determine el máximo número de caracteres que se podrán ahorrar en la impresión.

**Input**

La primera línea contiene un número entero T > 0 que denota el número de casos de prueba. Cada caso de prueba comienza con un número entero 2 ≤ N ≤ 100 que denota la cantidad de números telefónicos de la lista. Las N líneas siguientes contienen, cada una de ellas, un número telefónico de hasta 20 dígitos. Todos los números telefónicos de un mismo caso de prueba tienen la misma cantidad de dígitos. Un número telefónico puede comenzar con 0.

**Output**

Por cada caso de prueba, imprime una línea informando el mayor número posible de dígitos que se podrán ahorrar en la impresión.

**Sample Input**

2

2

12345

12354

3

535456

535488

835456

**Sample Output**

3

4

# Los caprichos del chef

Cierto reconocido chef de un anime se dedica a cocinar menús por tiempos. Sea que se trate de una cena en cuatro tiempos o de una comida en dos, se asegura siempre de que el orden en que los platillos se comen sea el óptimo para el disfrute de su comida. En realidad, a veces sus menús están compuestos por decenas de platillos. Para un menú específico junto a sus restricciones, usted debe calcular cuántas maneras posibles hay de ordenar los platillos satisfaciendo el criterio del chef.

**Entrada**

Cada caso de prueba se compone por una línea con un entero N (2<=N<=100) y un entero M (1<=M<=300) que indican respectivamente el número de platillos y el número de restricciones. Las siguientes N líneas se componen cada una de una cadena de caracteres alfabéticos (alfabeto inglés) representando el nombre de cada platillo. Las últimas M líneas representan las restricciones. Cada una está compuesta por los nombres de dos platillos separados por un único espacio. Cada restricción significa que el segundo platillo puede ser comido inmediatamente después del primero (“requisito”). Ningún platillo podrá ser comido si no es inmediatamente después de alguno de los platillos que tiene como requisitos, salvo aquellos platillos que no tengan requisitos. Hay que considerar que posiblemente habrá maneras de ordenar los platillos que no incluyan todos los platillos listados por el chef.

**Salida**

Por cada caso de prueba, una línea (terminada en salto de línea) incluyendo un único entero que represente el total de maneras posibles de ordenar los platillos satisfaciendo los caprichos del chef.

|  |  |
| --- | --- |
| **Entrada de muestra 1**  4 3  SOPITADEFIDEOS  ARROZ  POLLITOENCACAHUATADO  JERICALLA  SOPITADEFIDEOS POLLITOENCACAHUATADO  ARROZ JERICALLA  POLLITOENCACAHUATADO JERICALLA | **Salida de muestra 1**  2 |
| **Entrada de muestra 2**  8 7  Burritas  Sushi  Pastel  Teriyaki  OmeletteAuFromage  TortillaDePatata  Flan  ArrozConLeche  OmeletteAuFromage Burritas  TortillaDePatata Burritas  Burritas Sushi  Sushi Pastel  Sushi ArrozConLeche  Sushi Flan  Teriyaki Flan | **Salida de muestra 2**  7 |

# Minúsculas

Tienes un amigo interesado en las palabras que tienen la propiedad de que sus letras están escritas en orden alfabético, tales como: *adiós*, *ajo*, *himno*. Asimismo, le interesa saber si al combinar dos o más de estas palabras se forma alguna palabra interesante que siga estando en orden alfabético; sin embargo, la combinación puede ser muy cansado. Por tanto, tu amigo te pide ayuda para que le hagas un programa que reciba una lista de palabras con esta propiedad, y que imprima el resultado de combinar todas las palabras sin perder la propiedad. Como sabe que este problema es muy fácil para ti, también te pide que imprimas el resultado de combinar cada par de palabras adyacentes de la lista original, y cada par de combinaciones adyacentes sucesivas, hasta llegar a la palabra final. La restricción que le pones a tu amigo es que el número de palabras de la lista sea potencia de 2.

**Input**

La primera línea contiene un número entero T > 0 que denota el número de casos de prueba. Cada caso de prueba comienza con un número entero N ≥ 2 que denota el número de palabras. Se cumple que N es una potencia de dos. Le sigue una lista de N palabras formadas por 1 o más letras minúsculas acomodadas en orden alfabético y que se pueden repetir.

**Output**

Por cada caso de prueba, imprime en líneas separadas el resultado de combinar cada par de palabras adyacentes de la entrada. Luego, imprime el resultado de combinar cada par de palabras adyacentes que fueron combinadas en el paso anterior; y así, hasta llegar a una palabra que combina todo. Separa cada caso de prueba con una línea en blanco.

**Sample Input**

2

2

ace

bdf

4

adiooos

chinos

himno

ajo

**Sample Output**

abcdef

acdhiinooooss

ahijmnoo

aacdhhiiijmnnooooooss

# Multinivel

En una organización *multinivel*, los asociados son retribuidos no solo por las ventas que ellos mismos generan, sino también por las ventas generadas por las personas que forman parte del organigrama que cada uno encabeza. Es de esperarse que el individuo colocado en la parte más alta del organigrama (pirámide) tiene retribución por las ventas de todos los asociados.

La empresa FiruLife te ha contratado para que programes un algoritmo que calcule la ganancia total de su CEO y fundador, quien se encuentra colocado en la punta de su pirámide. En su estructura, cada asociado tiene cero, uno o dos asociados que le reportan sus ventas. A su vez, cada uno de ellos puede tener cero, uno o dos asociados. Cada asociado sólo conserva el 50% de sus ventas, el restante se distribuirá entre todas las personas de niveles superiores. De esta manera, el CEO se queda con el 50% de las ventas de los dos asociados que le reportan directamente a él (segundo nivel), 25% de los asociados de tercer nivel, 12.5% del cuarto nivel, y así sucesivamente. ¿Cómo se va formando esta pirámide binaria? El primer individuo que se incorpora a la organización será hijo directo del CEO. Cada nuevo asociado se incorpora al organigrama siguiendo la política de adición en un árbol binario de búsqueda: si su nombre es alfabéticamente menor que el del asociado actual, se incorpora al sub- organigrama izquierdo, si no, al derecho. Para mejor entendimiento, considere el siguiente ejemplo. El CEO se llama Jorge; los asociados se incorporaron en este orden: Higuera, Matías, Paco. En la figura se aprecia el organigrama resultante y el total de ventas de cada asociado en el último período.

Jorge ($50.00)

Higuera ($70.00)

Matías ($100.00)

Paco ($40.00)

En este caso, la ganancia total del CEO se calcula así: 50 + (0.5) 70 + (0.5) 100 + (0.25) 40 = **$145.00**.

**Input**

La primera línea contiene un número entero T > 0 que denota el número de casos de prueba. Cada caso comienza con un entero N que denota el número de asociados (0 < N < 100). Cada una de las N líneas siguientes comienza con una cadena de texto que denota el nombre del asociado, seguida de un espacio y un número real V que representa las ventas de tal asociado (0 < V < 1000). El primer asociado mostrado representa el CEO.

**Output**

La salida consiste en T líneas. Cada una contiene un número real con dos dígitos de precisión que representa las ventas totales del CEO por cada caso de prueba.

**Sample Input**

2

4

Jorge 50

Higuera 70

Matias 100

Francisco 40

8

Hugo 6.8

Paco 10.3

Luis 15

Donald 20

Pluto 18

Mickey 8.5

Daisy 30

Goofy 9.3

**Sample Output**

145.00

41.09

# Noticias falsas

Estás a cargo de la gestión de la difusión de noticias falsas de interés nacional a un grupo importante de suscriptores desempleados. La difusión de una noticia se tiene que realizar de boca en boca y tiene que llegar a todos los sucriptores. Cada uno tiene comunicación con uno o más suscriptores. Cada vez que un suscriptor *s*1 comunica una noticia falsa a un suscriptor *s*2, le ocasiona un grado de culpa *c*12, el cual es el mismo si la comunicación fuera en sentido contrario. Tu trabajo consiste entonces en difundir una noticia falsa a todos los suscriptores con el menor grado de culpa.

**Input**

La primera línea contiene un número entero *T* > 0 que denota el número de casos de prueba. Cada caso de prueba comienza con una línea que contiene un par de números enteros positivos *N*, *M* que denotan, respectivamente, el número de suscriptores y el número de grados de culpabilidad a capturar. Le sigue una línea que contiene *N* cadenas de texto separadas por un espacio: cada una está formada de letras mayúsculas o minúsculas y denota el nombre de un suscriptor. Siguen *M* líneas, cada una contiene dos nombres de suscriptores separados por espacio y el grado de culpabilidad cuando se comunican una noticia falsa, el cual es número real positivo. El caso de prueba finaliza con el nombre de un suscriptor quien será el que comience la difusión de la noticia falsa.

**Output**

Por cada caso de prueba deberás imprimir una línea que contiene la leyenda *“Culpabilidad total: X”*, tal que *X* denota la suma mínima de culpas que se obtiene tras difundir la noticia desde el suscriptor inicial a todos los demás, y se imprime con una decimal. La solución de este problema deberá inspirarse en el algoritmo de *Prim*. Por lo tanto, previo a la leyenda anterior, deberás imprimir una línea con el estado de la cola de prioridad cada vez que exista una operación *decreaseKey*, la cual sucede cuando un vecino haya disminuido su peso (grado de culpa). El formato de impresión de la cola se puede apreciar sin ambigüedad en el **Sample Output**. Cabe mencionar que, si un vértice aún no es alcanzable, se imprime *Inf*. Añade una línea en blanco al final de cada caso de prueba.

**Sample Input**

2

3 3

HUGO PACO LUIS

HUGO PACO 1.178

PACO LUIS 4.387

HUGO LUIS 2.143

HUGO

5 8

A B C D E

A B 56.73

B C 28.91

A C 65.73

A D 38.18

B D 87.12

C D 42.15

B E 18.65

C E 23.56

B

**Sample Output**

PACO (1.2), LUIS (Inf)

PACO (1.2), LUIS (2.1)

Culpabilidad total: 3.3

A (56.7), E (Inf), C (Inf), D (Inf)

C (28.9), E (Inf), A (56.7), D (Inf)

C (28.9), D (87.1), A (56.7), E (Inf)

E (18.7), C (28.9), A (56.7), D (87.1)

C (23.6), D (87.1), A (56.7)

A (56.7), D (42.2)

D (38.2)

Culpabilidad total: 137.1

# Permutación común

Dadas dos cadenas *a* y *b*, imprime la cadena más larga *x* de letras tal que hay una permutación de *x* que es subsecuencia de *a* y hay una permutación de *x* que es subsecuencia de *b*. Una cadena *x* es subsecuencia de una cadena *s* si y sólo si todos los caracteres de *x* están en ese mismo orden en *s*. Por ejemplo, “abc” y “ay” son subsecuencias de “axbyc”. Por lo anterior, toda subcadena es una subsecuencia, pero no toda subsecuencia es una subcadena.

**Input**

La entrada tiene un número no explicitado de casos, pero tiene al menos un uno. Cada caso consiste en dos líneas consecutivas. Esto significa que las líneas 1 y 2 son un un caso de prueba, las líneas 3 y 4 son otro caso de prueba, y así sucesivamente.

Cada línea contiene una cadena de caracteres del alfabeto inglés en minúsculas. La primera línea en un par es denotada por *a* y la segunda por *b*. Cada línea consiste de como mínimo 1 caracter y como máximo 1000.

**Output**

Por cada caso de prueba, despliega una línea (terminada en salto de línea) conteniendo a *x*. Si hubiera más de un *x* que satisficiera los criterios de arriba, elige el primero según el orden alfabético.

**Sample Input**

pretty

women

walking

down

the

street

**Sample Output**

e

nw

et

# Productos más vendidos

Una empresa está interesada en contratar tus servicios de *data mining*. La primera pregunta de negocio que le gustaría contestar es cuáles son los 3 productos más vendidos en un intervalo de tiempo. Para ello, la empresa te proporcionará un listado con los códigos de todos los productos involucrados en las transacciones de ventas que suceden en todas sus sucursales durante las últimas horas. Tu trabajo es encontrar los 3 códigos que aparecen más veces que los demás. Suele haber empates; en tal caso, deberás dar prioridad al producto con código menor.

**Input**

La primera línea contiene un número entero T > 0 que denota el número de casos de prueba. Cada caso de prueba está descrito en un renglón. Comienza con un número entero N, 3 ≤ *N* ≤ 10000, que denota el número de transacciones de ventas, seguido por N códigos de productos separados por espacio, tal que cada código es un número entero en el rango [0...9999].

**Output**

Para cada caso de prueba imprime tres números enteros A, B, C separados por espacio, que representan respectivamente el código del primer, segundo y tercer producto más vendidos, considerando el criterio de desempate descrito en el primer párrafo.

**Sample Input**

2

5 120 120 8 8 4

9 1500 88 178 1500 88 1500 964 178 1500

**Sample Output**

8 120 4

1500 88 178

# Que no se te vaya el tren

A una compañía ferroviaria le interesa automatizar el proceso que intercambia vagones de un tren, de forma que se coloquen del más corto al más largo. La máquina que efectúa cambios de posición de un vagón sólo puede efectuar intercambios entre dos vagones adyacentes. Tu misión es crear un programa que calcule el número mínimo de intercambios necesario para acomodar los vagones del tren del más corto al más largo.

**Input**

La entrada contiene en su primera línea el número de casos de prueba (T). Cada caso de prueba consiste de dos líneas. La primera línea contiene un entero Nque determina la longitud del tren o número de vagones, tal que: 0 ≤ N ≤ 60. La segunda línea contiene una permutación de los números 1 a N, indicando el orden actual de los vagones. Los vagones deberían ordenarse de tal forma que el vagón 1 aparezca primero, luego el 2, etc., terminando con el vagón N.

**Output**

Para cada caso de prueba imprime el enunciado: ‘Cantidad mínima de intercambios: *S*.’, donde *S* es un número entero no negativo. No te olvides del punto final.

**Sample Input**

2

3

1 3 2

4

4 3 2 1

**Sample Output**

Cantidad mínima de intercambios: 1.

Cantidad mínima de intercambios: 6.

# Ruta crítica

Para obtener la ruta crítica de un proyecto (entendido como una serie de actividades con duración, antecesoras y sucesoras) es necesario determinar qué actividades hay que terminar antes de poder empezar cada otra actividad. Ese procedimiento tiene algo de similar con otros. Por ejemplo, si consideramos que todas las actividades de un proyecto serán realizadas por la misma persona (y asumimos que una persona sólo puede realizar una actividad a la vez), hay que calcular una secuencia de actividades que esa persona debe llevar a cabo para terminar el proyecto, siguiendo con las restricciones de las actividades antecesoras y sucesoras. Cierto administrador muy metódico (que hace eco de cierto profesor muy pedagógico…) diría que si hay que decidir qué considerar primero entre varias actividades (posibles, claro), se elija siempre la actividad que fue planificada primero. Toda actividad A estará identificada por su tiempo de planificación (único) 1 <= t <= N, donde N es la cantidad total de actividades.

¡Ayude al administrador a calcular la secuencia!

**Input**

La primera línea contiene un número entero T > 0 que denota el número de casos de prueba. Cada caso de prueba comienza con una línea que contiene un par de números enteros N, M (2 <= N <= 1000, 1 <= M <= (N2 – N) / 2. Le siguen M líneas, cada una con dos enteros A y B (1 <= A <= N, 1 <= B <= N) que implican que la actividad B no puede ser iniciada antes de que la actividad A termine.

**Output**

Por cada caso de prueba, imprima una línea que contenga la lista de actividades en la secuencia. Separe las actividades entre sí utilizando un espacio.

**Sample Input**

1

9 9

5 6

6 7

5 7

9 7

6 1

2 1

2 3

3 4

1 4

**Sample Output**

9 8 5 6 7 2 3 1 4

# Secuencias

La bioinformática pretende aplicar las tecnologías de la información a la solución de problemas relacionadas con el manejo de datos biológicos. Un dato biológico que ha cobrado mucho interés en investigaciones recientes relacionadas con reconocimiento de patrones es la secuencia ADN.

Una secuencia de ADN es una sucesión de letras que representa la estructura primaria de una molécula real o hipotética de ADN con la capacidad de transportar información. Las posibles letras que puede contener esta cadena son A, C, G, y T, que simbolizan las cuatro subunidades de nucleótidos: adenina, citosina, guanina, timina, las cuales son bases covalentemente ligadas a cadenas fosfóricas.

Para muchos seres vivos, la longitud de la secuencia ADN que define su genoma se encuentra en el orden de los miles. Esto obliga a implementar mecanismos inteligentes para realizar de manera eficiente operaciones con secuencias ADN, como: búsquedas y ordenamientos. Un mecanismo que puede acelerar la búsqueda de una secuencia ADN en una lista grande es construir un número entero a partir de realizar operaciones aritméticas con las letras que la constituyen, luego almacenar la secuencia en la posición de una lista indicada por dicho número entero. Para obtener el número entero se utiliza la fórmula del cambio de base, tal que la base en este contexto es 4 (A, C, G, T):

Tal que denota el valor numérico del caracter en la posición de la secuencia (de izquierda a derecha). Los valores numéricos de A, C, G y T son, respectivamente, 1, 2, 3 y 4.

Por ejemplo,

Debido a que el resultado puede ser un número entero demasiado grande, se calcula el residuo de dividir el resultado entre algún número primo.

**Input**

La primera línea contiene un número entero T > 0 que denota el número de casos de prueba. Cada caso de prueba comienza con una línea que contiene un par de números enteros N, M (2 ≤ N ≤ 100, 10 < M < 200000) que denotan, respectivamente, la cantidad de secuencias ADN a procesar y el número primo a usar para calcular el residuo. Le siguen N líneas, cada una contiene una secuencia ADN de longitud 2 ≤ L ≤ 20 conformada por las letras mayúsculas A, C, G, T.

**Output**

Por cada caso de prueba deberás imprimir una línea que contiene las secuencias ADN separadas con espacios y ordenadas de menor a mayor de acuerdo al número entero correspondiente que se describió anteriormente. Si el tamaño de la lista de secuencias es mayor que cinco, imprime sólo las primeras cinco secuencias. Dos o más secuencias que les corresponde el mismo número entero deberán ordenarse de manera alfabética.

**Sample Input**

2

4 7

CG

ACG

AACG

AACT

4 17

CG

ACG

AACG

AACT

**Sample Output**

CG ACG AACG AACT

AACT ACG AACG CG

# Subsecuencias distintas

Una subsecuencia de una secuencia dada S consiste en S con cero o más elementos eliminados. Formalmente, una secuencia *Z* = *z*1*z*2…*zk* es una subsecuencia de *X* = *x*1*x*2…*xm* si existe una secuencia estrictamente creciente *i*1, *i*2..., *ik* de índices de *X* tal que para todo *j* = 1, 2, . . . , *k*, tenemos *xi*[*j*] = *zj*. Por ejemplo, *Z* = *bcdb* es una subsecuencia de *X* = *abcbdab* con la correspondiente secuencia de índices: 2, 3, 5, 7. Tu trabajo es escribir un programa que cuente el número de ocurrencias de *Z* en *X* como una subsecuencia, tal que cada una tiene una secuencia de índices distinta.

**Input**

La primera línea contiene un entero *N* indicando el número de casos de prueba. La primera línea de cada caso de prueba contiene una cadena *X*, compuesta enteramente de caracteres alfabéticos en minúscula y con una longitud no mayor que 10,000. La segunda línea contiene otra cadena Z de longitud no mayor que 100 y también compuesta sólo de caracteres alfabéticos en minúscula.

**Output**

Por cada caso de prueba, muestra el número de ocurrencias distintas de *Z* en *X* como una subsecuencia. La salida de cada conjunto de entrada debe estar en una línea separada (terminada con salto de línea).

**Sample Input**

2

babgbag

bag

rabbbit

rabbit

**Sample Output**

5

3

# Sucursales convenientes

Una importante cadena de tiendas de conveniencia tiene muchas sucursales en varias ciudades y pueblos del estado. Además, tiene registrado el tiempo promedio que tarda cada uno de sus auditores en trasladarse desde cada sucursal hacia algunas de las demás. Este registro le permite tomar decisiones más certeras acerca de la ruta que le conviene seguir a un auditor para visitar la mayor cantidad de sucursales en una misma jornada de trabajo. Debido a que no tiene todos los tiempos registrados, pueden existir rutas aún más rápidas. Para reducir esta posibilidad, el gerente general decide invertir en algunos mensajeros que calculen el tiempo de traslado entre *E* pares de sucursales no registradas antes. Luego te pide a tí que calcules si las rutas más cortas fueron mejoradas o no.

**Input**

Cada caso de prueba comienza con dos enteros positivos *S*, *D* separados por espacio. *S* denota el número de sucursales y *D* el número de distancias entre cada par de sucursales registradas. Le siguen *D* líneas, cada una con tres números enteros separados por espacio. Los primeros dos *s*1, *s*2 denotan el identificador de la sucursal tal que: 1 ≤ *s*1, *s*2 ≤ *S*. El tercer número es el tiempo promedio de viaje para llegar de *s*1 a *s*2 y viceversa. Consideramos que el tiempo de ida es igual al de vuelta. Le sigue un número entero *E* > 0 que denota el número de tiempos de traslado que se añaden al registro. Le siguen *E* líneas, cada una denotando el tiempo de traslado entre dos distintas sucursales. Después de leer un caso de prueba, -1 significa que no hay más casos de prueba (-1 no será procesado).

**Output**

Por cada caso de prueba, imprime *No se ahorró tiempo* si al añadir los nuevos *E* tiempos de traslado, la suma de los tiempos de las rutas más cortas no cambió. En otro caso, imprime *Ahorro total: X*, tal que *X* denota la reducción en el tiempo total, es decir, la suma del tiempo de la ruta más corta entre cada par de sucursales.

**Sample Input**

3 2

1 2 10

2 3 15

1

1 3 26

4 3

1 2 3

2 3 4

3 4 2

2

1 3 6

2 4 6

-1

**Sample Output**

No se ahorró tiempo

Ahorro total: 2

# Utilidades

Estás interesado en iniciar un pequeño negocio de ventas de frutas, verduras y otros alimentos dentro del prestigioso y alejado fraccionamiento donde vives. Para ello, te surtes en los mercados del oriente de la ciudad en donde encuentras todo a precios bastante accesibles. Se te presenta un catálogo muy variado de productos, cada uno con su precio de compra y el precio sugerido de venta. Sin embargo, cada domingo que vas al mercado cuentas con un presupuesto limitado para adquisición de tus insumos. Estás seriamente interesado en comprar sólo aquellos productos que, en suma, te arrojen la mayor utilidad, sin rebasar tu presupuesto. Para mayor entendimiento del problema, considérese el siguiente ejemplo. El catálogo del día consta de 8 productos: A, B, …, H. La primera fila corresponde al precio de compra, mientras que la segunda fila al precio de venta sugerido.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D | E | F | G | H |
| $ 100.00 | $ 50.00 | $ 70.00 | $ 220.00 | $ 90.00 | $ 50.00 | $ 260.00 | $ 60.00 |
| $ 150.00 | $ 100.00 | $ 150.00 | $ 300.00 | $ 110.00 | $ 70.00 | $ 300.00 | $ 100.00 |

Suponiendo que el presupuesto del día es $500.00. El producto con la menor relación compra/venta es C. Entonces si seleccionamos los productos C, B, H, A, F, tendremos un gasto de $330.00 y una ganancia de $240.00. Aún contamos con $170.00. El siguiente producto elegible será el D. Como estos productos se venden por kilo, podemos comprar una fracción de él. En este caso, compramos 170/220 de D, obteniendo una ganancia total máxima de $301.82.

**Input**

La primera línea contiene un número entero T > 0 que denota el número de casos de prueba. Cada caso de prueba comienza con un número real P > 0 que denota el presupuesto. Después un número entero N > 0 que denota el número de productos del catálogo. Le siguen N líneas con dos números reales separados por espacios que representan, respectivamente, precio de compra y precio de venta.

**Output**

Por cada caso de prueba, imprime la utilidad máxima que se puede obtener con el criterio de selección mencionado arriba, utilizando dos decimales de precisión.

**Sample Input**

2

500

8

100 150

50 100

70 150

220 300

90 110

50 70

260 300

60 100

100

4

74.5 103.6

25.8 51.7

51.235 60.83

60 90.3

**Sample Output**

301.82

61.75

# Un tercer cuerpo

En la filosofía de Baruch Spinoza se dice que dos cuerpos componen sus relaciones cuando aumentan mutuamente sus *potencias* (lo que son capaces de hacer: lo que *pueden* hacer). Para él esa composición de relaciones es a su vez un cuerpo, de forma que para Spinoza el amor da lugar a un *tercer cuerpo*. Dada una lista de números enteros y un número *X*, determine si *X* es el tercer cuerpo formado por la suma de dos números en la lista. [[2]](#footnote-2) Considere que un elemento de la lista no puede sumarse consigo mismo, aunque sí con otro elemento que tenga el mismo valor.

**Input**

Este problema tiene un número no explicitado de casos de prueba. Cada caso está compuesto como sigue. La primera línea contiene al número *X*. La siguiente línea contiene *N*, que es el tamaño de la lista. Las siguientes *N* líneas contienen cada una un elemento *i* de la lista, donde un elemento *i* nunca será menor que 0 ni mayor que *N*.

**Output**

Por cada caso de prueba, imprima una línea que contenga la palabra *yes* o la palabra *no* seguida de un salto de línea, dependiendo de si hubo un par de elementos en la lista que sumara exactamente *X*.

|  |  |
| --- | --- |
| **Sample Input**  5  4  3  2  3  1  8  5  4  2  5  1  2 | **Sample Output**  yes  no |

1. En el uso de la lengua castellana no hay distinción entre el fonema representado por ‘b’ y el fonema representado por ‘v’. Estaciones de ser vicio suena exactamente igual que estaciones de ser bicio. [↑](#footnote-ref-1)
2. Para ser más precisos, Spinoza no podría ver la composición de relaciones como una suma, sino como una sinergia. Como cuando se dice que el todo es más que la suma de las partes. [↑](#footnote-ref-2)