Cuarta maratón de programación de los cursos de Algoritmos.
Categoría Borg

Instrucciones

1. URL del juez: https://acm.javeriana.edu.co/maratones/2016/Borg/

Maratón de Programación Interna de Algoritmos
Es un espacio de sana competencia donde los estudiantes de los cursos de algoritmos trabajan en equipos para resolver la mayor cantidad de ejercicios en el menor tiempo posible.

Los ejercicios propuestos en la maratón combinan pensamiento lógico, habilidades algorítmicas y deben ser resueltos a través de un programa de computador escrito en lenguaje Java bajo unas condiciones especificadas desde el inicio de la competencia.

Cada equipo estará conformado por máximo dos estudiantes quienes deben elegir un nombre que los identifique para registrarse en la maratón. El evento cuenta con tres categorías cada una con su propio conjunto de problemas y orientada a un grupo particular de estudiantes.

La participación de los estudiantes es en equipos de dos personas. Cada equipo debe tener un nombre que lo identifique en la maratón y debe ser definido en el momento de la inscripción. La organización de la competencia entregará premiación a los tres primeros equipos de cada categoría. La participación en la maratón no es obligatorio y, debido a restricciones de espacio, los cupos son limitados.

Reglamento:

1. SEDE Y FECHA
La maratón interna de algoritmos de la universidad Icesi se realiza en la universidad Icesi es las salas de cómputo 202C y 203C. Para el primer semestre de 2016, ésta se realizará el Viernes 13 de Mayo de 2016.
2. CATEGORÍAS
La maratón interna se desarrollará en 3 categorías, en ellas podrán participar estudiantes que esté cursando a lo sumo el curso descrito en la categoría, estas son:

- **Categoría Dijkstra**. Para los estudiantes de Algoritmos y Programación I - Fundamentos de programación en diseño.

- **Categoría Turing**. Para los estudiantes de Algoritmos y Programación II - Diseñando con Algoritmos.
- **Categoría Borg**. Para los estudiantes de Algoritmos y Estructuras de Datos y semestres posteriores.

En cuanto a restricciones de género no hay ninguna, todas las categorías son abiertas por lo que los equipos pueden ser 2 personas del mismo género, o mixto.

3. REGISTRO DE PARTICIPACIÓN

Podrán participar los estudiantes de la universidad Icesi inscribiéndose por medio de un formulario web que se les hará llegar via correo electrónico.

4. MÍNIMOS PARA LA REALIZACIÓN DE LAS COMPETENCIAS

Para que exista competencia el número mínimo deberá ser de cinco (5) equipos en la categoría correspondiente.

5. PREMIACIÓN

En la maratón interna de Icesi se estarán disputando, 3 Medallas de oro, 3 medallas de plata y 3 medallas de bronce. Las cuales serán distribuidas 1 por categoría.

La premiación oficial para la maratón interna de algoritmos es la siguiente:

1er Puesto: Medalla Dorada, Bono Ventolini, Matrícula Preferencial
2do Puesto: Medalla Plateada, Bono Cafetería Icesi
3er Puesto: Medalla de Bronce.

6. CONDICIONES PARA SER ACREEDOR A LA PREMIACIÓN

La ‘‘Matrícula Preferencial’’ es la ayuda que le brinda la universidad Icesi a las personas que la representan en competencias, la cual consiste en dar prioridad durante la semana de matrículas a los deportistas, para que de esta manera puedan matricular los cursos de su semestre sin afectar sus horarios de entreno. Asi es como se incentivan a los deportistas a seguir participando por la universidad en los diferentes torneos que se realizan durante la jornada académica.

Ya que este reconocimiento es uno de los más grandes que se puede dar a nivel académico, para que el estudiante sea acreedor de ella debe cumplir las siguientes condiciones:

A. Obtener el primer puesto en la categoría inscrita.
B. Su equipo debe haber resuelto al menos un problema en la maratón.
C. No puede ser repitente del curso
D. No puede haber cancelado el curso en semestres anteriores.
E. No puede estar en prueba académica.
F. No puede tener procesos disciplinarios pendientes.

G. Debe asistir el próximo semestre al club de programación.

Tabla de Contenido

Problema A: Tráfico en Ilac

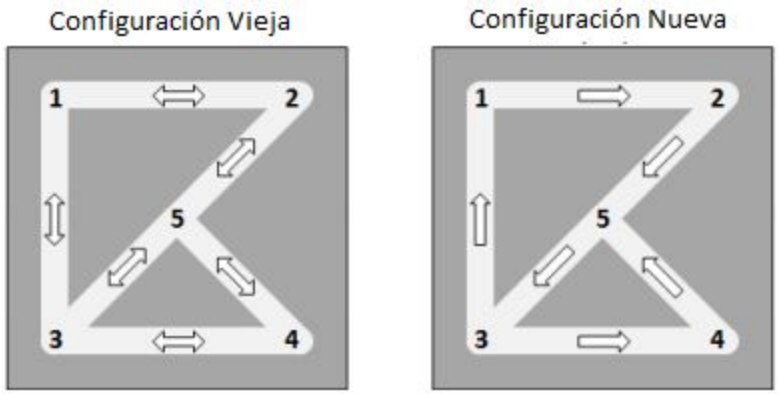
Base Name: trafico.java

Autor: Camilo Barrios

Ilac es una ciudad que se encuentra totalmente comunicada. Eso significa que los habitantes de Ilac han decidido que debe ser posible viajar desde cualquier punto de la ciudad a cualquier otro punto usando un automóvil. Cada punto en Ilac está situado en una cuadra, y cada cuadra se conecta a dos intersecciones (de modo que no hay intersecciones entre una calle y otra, y que no hay más de una calle que conecte dos intersecciones). Las autoridades de la ciudad como parte de su plan de movilidad siempre han garantizado que la ciudad es conexa, es decir: Que desde cualquier intersección de la ciudad se puede llegar a cualquier otra. Para lograr esto las autoridades tuvieron que dejar algunas calles que van en doble sentido. Dadas dos intersecciones X e Y en Lica, la distancia de X a Y se mide como el número mínimo de calles que deben ser recorridas para ir de X a Y.

Últimamente han habido trancones en varias calles de la ciudad y estos se presentan casi en todo momento. Los expertos recomiendan una solución sencilla: basta con cambiar algunos calles de doble sentido y dejarlas en un solo sentido. Sin embargo, está claro que estos cambios deben hacerse con cuidado, ya que la accesibilidad entre los puntos de la ciudad se puede perder. Incluso, si se garantiza la accesibilidad, es posible que las distancias entre intersecciones específicas puedan aumentar significativamente

Después de muchas discusiones, los asesores del alcalde han recomendado aceptar cualquier propuesta que aumenta la distancia entre dos intersecciones por un factor A aumentada por una constante B, con respecto a la configuración anterior (es decir, si la distancia actual de una intersección a otra es x, entonces la nueva distancia debe ser como máximo A · x + B). A continuación podemos ver una imagen de una configuración de calles de Ilac (Esta es la configuración presentada en el primer caso de prueba)



Tu estás contratado para desarrollar un programa que decida si una propuesta dada de la nueva dirección de las calles en la ciudad satisface los requerimientos.

Entrada

La entrada consta de varios casos de prueba, cada uno de los casos tiene un conjunto de líneas:

- La primera línea contiene un entero n ($3 \leq n \leq 100$) que representa el número de intersecciones en Lica. Supongamos que las intersecciones se identifican por números naturales en el conjunto {1,. . . , N}.
- A continuación vendrá una línea por cada una de las intersecciones. Esta línea tendrá un conjunto de números. El primer número hace referencia a la intersección se está describiendo en el momento (Nodo Actual). Los demás números son las intersecciones con las cuales se conecta por medio de una calle (Nodos Adyacentes). Todas estas líneas representan la configuración inicial (Vieja) de las calles de Ilca.
- Las siguientes n líneas describen, con el mismo formato ya se ha especificado, la nueva propuesta. Cabe aclarar que en la descripción de las calles, la dirección de estas va orientada hacia afuera, es decir, desde el primer elemento en la línea para cada uno de los elementos adyacentes (el mismo hecho se aplica a la configuración antigua).
- La descripción del caso termina con una línea con dos valores enteros A y B ($0 \leq A \leq 10$, $0 \leq B \leq 10$).

El último caso de prueba es seguido por una línea que contiene un solo 0.

Salida

Para cada caso de prueba, Se debe imprimir una salida con la palabra “Cumple!” En caso de que la nueva propuesta satisface todos los requerimientos o sin “No cumple :(” en caso contrario.

Ejemplo

Entrada	Salida
5 1 2 3 2 1 5 3 4 5 1 4 3 5 5 2 3 4 1 2 2 5 3 1 4 4 5 5 3 1 2 5 1 2 3 2 1 5 3 4 5 1 4 3 5	Cumple! No cumple :(Cumple!

5 2 3 4 1 2 2 5 3 1 4 4 5 5 3 2 0 3 1 2 2 1 3 3 1 2 1 2 2 3 3 1 0 2 0	

Problema B: Gran Hermano

Base Name: hermano.java
Autor: Juan Manuel Reyes - Universidad Icesi

El tiempo del gran omnipresente y vigilante Gran Hermano augurado por George Orwell ha llegado. Todos tienen un teléfono móvil, la mayor parte están controlados por la gran empresa de la G, luego sigue la de la A y luego la de la M. Ellos se han unido para formar el gran hermano cuyo nombre ahora es GAM. Tienen nuestros datos, tienen nuestras conversaciones, conocen nuestras vidas.

Todas las personas tienen un identificador y GAM lo conoce. Todo lo que haces a través de tu móvil ellos lo saben y están empezando a controlar los gobiernos de todos los países. No lo sabes, pero tu ADN ya está asociado con ese número y cada cosa que haces, cada lugar que visitas queda registrado en la gran computadora central.

Ellos te han contratado, no puedes resistirte porque sino serás eliminado. Dado un gran listado de identificaciones obtenidas de su gran base de datos y un listado de sospechosos de conspirar contra el régimen, usted debe establecer si el identificador del sospechoso se encuentra en la base de datos.

Entrada

La primera línea contiene un número entero $0<c\leq100$ con la cantidad de casos de prueba. Luego siguen c casos de prueba así: la primera línea de cada caso tiene dos números enteros $1\leq n\leq10^9$, $1\leq q\leq10^9$. El primero es la cantidad de identificadores en la base de datos y el segundo es la cantidad de identificadores de sospechosos a ser consultados. Luego siguen n identificadores, uno por línea, pertenecientes a la base de datos. Y luego q identificadores de sospechosos a buscar, uno por cada línea.

Salida

Por cada identificador consultado se imprime una línea con SI, si el identificador del sospechoso se encuentra en la base de datos y NO, en caso contrario. Entre las respuestas de cada caso de prueba hay una línea en blanco con un salto de línea.

Ejemplo

Entrada	**Salida**
2 5 2 KD367 VYJ75 PWQ74 EVY356 CVB21 HKF593 PWQ74	NO SI NO

3 1 ERTU347 GURYU356 XDF478 VYJ75	

Problema C: Máximo dos veces

basename: twice.java

En la competencia Latinoamericana de programación ACM/ICPC 2015, uno de los problemas era: hallar el número menor o igual a un número N, lo más grande posible, que tuviera **MAXIMO** dos repeticiones de cada digito.

Sus compañeros clasificados de la universidad Icesi no pudieron resolver este problema, y por eso le han pedido ayuda a usted, ya que han escuchado que usted es uno de los mejores programadores de la Universidad y desean que los ayude a entrenar para la próxima maratón latinoamericana.

Entrada

La primera línea de la entrada es un numero T < 100, indicando la cantidad de casos de prueba. Después siguen T líneas, cada una con un numero $1 \leq N \leq 10^{18}$

Salida

Usted deberá imprimir T líneas, cada una con el número menor o igual a N, lo más grande posible, que tengo máximo dos repeticiones de cada digito

Ejemplo

Entrada	Salida
4 1000000000000000000 20162016 2210102960 1001223343	998877665544332211 20162016 2210099887 998877665

Problema D: Problema del viajero intergaláctico

Base Name: viajero.java

Autor: Juan Sebastián Prada

Por favor no olvide su toalla antes de leer este problema.

El Motor de Improbabilidad Infinita es un método nuevo y fantástico para recorrer grandes distancias interestelares en segundos, sin tener que ir a tontas y a locas por el hiperespacio. En cuanto el motor de improbabilidad infinita alcanza la improbabilidad infinita, pasa por todos los puntos posibles de todos los universos posibles de forma casi simultánea. En otras palabras, nunca sabes dónde vas a ir a parar, ni siquiera qué especie serás cuando llegues, por tanto es importante ir bien vestido. El motor de improbabilidad infinita se inventó a partir de estudios sobre la improbabilidad finita, que se empleaba para romper el hielo en las fiestas, haciendo que las moléculas de la ropa interior de la anfitriona saltasen treinta centímetros a la izquierda, de acuerdo con la teoría de la indeterminación. Muchos físicos respetables mostraron su desaprobación, en parte porque constituía una degradación científica, pero principalmente porque no los invitaban a esa clase de fiestas.

El motor de improbabilidad infinita ha traído ciertos problemas cuando se trata de normalizar la especie de los viajeros. Esto debido a que existen ciertas especies incompatibles y existen transiciones forzadas durante el viaje. Afortunadamente es posible saber si volverás a tu especie original o al menos a una parecida.

Los grupos de especies determinan que especies son compatibles y así saber si es posible reversar el cambio causado por el motor de improbabilidad infinita.

Por ejemplo los siguientes grupos.

Humanoides: Humanos, Magratheanos.

Humanos: Simios, Delfines, Perros.

Vogones: Ratones, Betelgusianos.

Betelgusianos: Dentrassis.

Es decir si soy un humanoide y al terminar en una especie como Delfin podría fácilmente volver a la normalidad.

Su misión es determinar sí es posible regresar a la especie inicial o de lo contrario morir como una diferente.

Entrada

La primera línea contendrá dos números N y M, N será el número de grupos de especie y M el número de verificaciones de normalización que deberá realizar. Continúan N líneas con un numero K de Strings separados por espacios, K >0, en dónde el primer String será el nombre del grupo y los siguientes serán sus sub-grupos. Luego siguen M líneas con 2 Strings separados por espacios el primer String indica la especie en la que se terminó y el segundo String indica cual es su especie origen.

Salida

Por cada búsqueda deberá imprimir “42” en caso de que sea posible volver a la normalidad y “MUERTO” en caso de que no.

Ejemplo

Entrada	Salida
4 3 humanoides humanos magratheanos humanos simios delfines perros vogones ratones betelgusianos betelgusianos dentrassis humanos delfines vogones dentrassis betelgusianos humanos	42 42 MUERTO

Problema E: La Venganza de C

Nombre del Archivo: ancestro.java

Autor: Felipe Clement

En aquel día, predestinado quizás por los mismos dioses, M le pregunto a C: "¿Podrías encontrar el nodo ancestro común inferior entre dos nodos de un BST?", a lo cual C respondió con indecisión: "Meh, supongo". Pero C no lo hizo bien, no alcanzo a terminar y además de esto, ¡la solución que estaba desarrollando era aplicable a cualquier árbol binario!

C quedo traumatizado, ya no puede mirar un BST sin tener retrospectivas dolorosas de aquel momento, ¡pero él siempre está en la búsqueda de la verdad! Por lo tanto, le ha delegado a usted la tarea de resolver este problema para por fin obtener paz mental.

Entrada

La entrada contiene varios casos de prueba. Cada caso de prueba empieza con un número N (N > M) y los M (M < 50) nodos del árbol (siempre empiezan en cero, ascendentemente); N indica las N líneas siguientes del caso de prueba. Las N - 1 líneas siguientes consisten de tres números cada una, el primer número A es el valor de un nodo del árbol, los dos números siguientes B y C (B <= A y C > A) indican él valor de los hijos izquierdo y derecho respectivamente del nodo A, con la letra "n" denotando que el hijo es null. La última línea del caso especifica los dos nodos (garantizados a estar en el árbol) del árbol a los cuales se les debe encontrar el ancestro común inferior.

Salida

Por cada caso de prueba, debe imprimir el valor del nodo ancestro común inferior.

Ejemplo

Entrada	Salida
16,0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 7 3 11 3 1 4 11 10 12	10

1 0 2 4 5 6 10 8 9 13 12 14 0 n n 2 n n 5 n n 6 n n 8 n n 9 n n 12 n n 14 n n 8 10	

Problema F: Vértices inaccesibles

Base Name: vertices.java

Autor: Brayan Henao

Se le ha pedido a usted, un habilidoso estudiante, que escriba un programa que busque en un grafo dirigido todos los vértices que son inaccesibles desde un vértice dado.

Un grafo dirigido está representado por N vértices, donde $\mathbf{1} < \boldsymbol{n} \leq \mathbf{100}$, numerados consecutivamente **1 . . . n**, y una serie de aristas ***u*** --> ***v*** la cual conecta una un par de nodos ***u*** y ***v*** , en una sola dirección.

Un vértice **r** es accesible desde el vértice ***p*** si existe alguna arista ***p***--> ***r,*** o si existe algún vértice ***q*** para el cual ***q*** es accesible desde ***p*** y ***r*** es accesible desde ***p***.

Un vértice ***r*** es inaccesible desde el vértice ***p*** si ***r*** no es accesible desde el vértice ***u***.

Entrada

La entrada de este problema consiste en varios grafos dirigidos y nodos de inicio.

Para cada grafo, la primera línea contiene un entero N, el número de nodos en el grafo.

Siguiendo, habrá un grupo de líneas, cada una contiene una serie de enteros. El grupo de líneas es finalizado por una la línea que contiene solo el entero 0. Para cada línea, el primer entero, *i*, es el vértice de inicio. Los siguientes enteros, *j, k, …z*, definen una serie de aristas *i* --> *j, i* --> *k, ... i*-->*z*, y el último entero en la línea siempre será 0 (No debe ser procesado). Cada posible vértice de inicio *i*, $1 < i \leq n$.

Una vez proporcionada la información del grafo, seguirá una línea la cual contiene una lista de enteros. El primer entero en la lista será el número de enteros qué le siguen en esta, cada uno de estos enteros representa el vértice de inicio, el cual deberá ser investigado por su programa.

La línea siguiente contiene la información del próximo grafo. Si no hay más grafos, la línea del archivo contendrá sólo el entero 0.

Salida

Para cada vértice de inicio a ser investigado, su programa debe identificar todos los vértices que son inaccesibles desde ese vértice de inicio. Cada lista deberá aparecer en una línea, con el número de vértices que son inaccesibles y seguido, el número de cada vértice inaccesible.

Ejemplo

Entrada	Salida
3 1 2 0 2 2 0 3 1 2 0 0 2 1 2 5 0 5 1 2 3 4 5 0	2 1 3 2 1 3 5 1 2 3 4 5 5 1 2 3 4 5 5 1 2 3 4 5 5 1 2 3 4 5 5 1 2 3 4 5