Algoritmos y Estructuras de Datos 3 (Período II, 2019)

Instructor: Marcos Villagra
Auxiliar de cátedra: Julio Mello

Tarea #2
Agosto 15, 2019

Consideraciones para la Entrega

1. Por cada problema resuelto preparar un archivo `.java` para cada problema con el nombre del problema, por ejemplo, `problema1.java`, `problema2.java`, etc.

2. En cada archivo `.java` incluir ahí el código fuente. Utilizar solamente un archivo `.java` por problema. No separar en archivos distintos diferentes partes del código fuente. Todo el código que corresponde a un problema debe de estar en un solo archivo. Todas las clases en este archivo no deben estar dentro de ningún paquete. Todos los programas deben comenzar en un método principal estático en una clase `Main` (`class Main`). No utilice clases públicas: incluso `Main` no debe ser público para evitar errores de compilación. Utilice las E/S estandard para evitar que se exceda el límite de tiempo. Todas las líneas terminan con un caracter de fin de línea '\n'. Ver ejemplo en Anexo 1 al final de este documento.

3. Preparar un archivo de texto aparte con el nombre `grupo.txt` en donde se indique los integrantes del grupo con sus nombres completos. Este archivo puede escribirse en cualquier formato.

4. Comprimir todos los archivos juntos en un `.zip` o `.rar` y cargar ese único archivo en el enlace habilitado en Educa.

5. Los archivos que no esten preparados siguiendo las indicaciones de los puntos 1, 2, 3 y 4 arriba citados, no serán considerados para evaluación.

6. Todos los código fuentes recibidos serán evaluados por un *juez en línea* que verificará la correctitud de la solución y si se cumplen las restricciones de tiempo. Por lo tanto, el input y el output de cada programa deben hacerse por *standard input* y *standard output* y seguir estrictamente el formato de entrada y salida establecido por cada problema.

7. Cada grupo debe de asegurarse que el input y el output especificados para cada problema sean los correctos. Si no se respeta ese formato, el juez rechazará el código con error y ese problema no será puntuado.

8. Los problemas que contienen errores de formateo de input o output, de compilación o fallan en algún caso especial no llevaran puntos.

9. El problema que es aceptado por el juez llevará la nota completa. En el caso en que el juez entregue un error `TLE` (*time limit exceeded*) y los tiempos no son excesivamente altos, se pasará a la verificación manual del código para verificar la implementación correcta.

10. Todos los códigos entregados serán verificados en búsqueda de plagios por métodos manuales y automáticos.

11. Tener en cuenta las consideraciones explicadas el primer día de clase para la presentación de trabajos prácticos. Puedes volver a consultar esa información en `http://www.cc.pol.una.py/~mvillagra/courses/aed3-2019/algoritmos3-2019-reportes.html`.

Problema (1) Ordenamiento infrecuente.

Límite de tiempo: 3 segundos

Un coleccionista de libros raros descubrió recientemente un libro escrito en un idioma desconocido que usaba los mismos caracteres que el idioma inglés. El libro contenía un índice corto, pero el orden de los elementos en el índice era diferente de lo que cabría esperar si los caracteres se ordenaran de la misma manera que en el alfabeto inglés. El coleccionista intentó usar el índice para determinar el orden de los caracteres (es decir, la secuencia de clasificación) del alfabeto extraño, luego se rindió con frustración ante el tedio de la tarea.

Debes escribir un programa para completar el trabajo del coleccionista. En particular, su programa tomará un conjunto de cadenas que se han ordenado de acuerdo con una secuencia de clasificación particular y determinará cuál es esa secuencia.

Entrada

La entrada consiste en una lista ordenada de cadenas de letras mayúsculas, una cadena por línea. Cada cadena contiene como máximo 20 caracteres. El final de la lista está señalado por una línea con el carácter único '#'. No todas las letras se usan necesariamente, pero la lista implicará un orden completo entre las letras que se usan.

Salida

Su salida debe ser una sola línea que contenga letras mayúsculas en el orden que especifique la secuencia de clasificación utilizada para producir el archivo de datos de entrada.

Ejemplo de entrada
XWY
ZX
ZXY
ZXW
YWWX
#

Ejemplo de salida
XZYW

Problema (2) Grafos conectados.

Límite de tiempo: 3 segundos

Considere un grafo G formado por una gran cantidad de nodos conectados por aristas. Se dice que G está conectado si se puede encontrar un camino en 0 o más pasos entre cualquier par de vértices en G.

Un subgrafo conectado es máximo (componente conectado) si no hay vértices y aristas en el grafo original que puedan agregarse al subgrafo y aún así dejarlo conectado.

Escriba un programa para determinar el número de subgrafos conectados máximos (componente conectado) de un grafo dado.

Entrada

La entrada comienza con un solo entero positivo en una línea que indica el número de casos siguientes, donde cada uno de ellos como se describe a continuación. A esta línea le sigue una línea en blanco, y también hay una línea en blanco entre dos entradas consecutivas.

La primera línea de cada conjunto de entrada contiene un solo carácter alfabético en mayúsculas. Este carácter representa el nombre del vértices de mayor valor en el grafo, siendo el vértice de menor valor siempre `A`. Cada línea sucesiva contiene un par de caracteres alfabéticos en mayúsculas que denotan una arista en el grafo.

El ejemplo de entrada contiene un posible conjunto de entrada. La entrada termina con una línea en blanco.

Salida

Para cada caso de prueba escriba en la salida el número máximo de subgrafos conectados. Las salidas de dos casos consecutivos estarán separadas por una línea en blanco.

Ejemplo de entrada

```
1

E
AB
CE
DB
EC
```

Ejemplo de salida

```
2
```

Problema (3) La Red.

Límite de tiempo: 3 segundos

Teniendo en cuenta el interés actual en Internet, un enrutamiento inteligente de información se convierte en una necesidad. El trabajo de enrutamiento de datos lo realizan enrutadores ubicados en los nodos de una red. Cada enrutador tiene su propia lista de enrutadores que son visibles para él (llamada "tabla de enrutamiento"). La información debe enrutarse de manera que minimice el número de enrutadores por la que tiene que pasar (lo que se denomina "conteo de saltos").

Su tarea es encontrar una ruta óptima (de número mínimo de saltos) para la red dada desde la fuente de la información hasta su destino.

Entrada

La primera línea contiene el número (n) de enrutadores en la red. Las siguientes n líneas contienen una descripción de la red. Cada línea contiene un ID de enrutador, seguido de un guión y una lista separada por comas de ID de enrutadores visibles. La lista está ordenada en orden ascendente.

La siguiente línea contiene (m) números de rutas que debe determinar. Las m líneas consecutivas contienen enrutadores iniciales y finales para la ruta separados por un solo espacio. Los datos de entrada pueden contener descripciones de muchas redes.

Salida

Para cada red, debe generar una línea con 5 guiones y luego, para cada ruta, una lista de enrutadores por los que pasó la información enviada desde el inicio hasta los enrutadores de destino.

En caso de que sea imposible pasar información (no existe conexión), debe generar una cadena "`connection imposible`". En el caso de múltiples rutas con el mismo conteo de saltos, se debe generar el que tenga ID más bajos (en el caso de la ruta del enrutador 1 a 2 como "`1 3 2`" y `1 4 2`", debe aparecer "`1 3 2`" en la salida).

Suposiciones: el número de enrutadores no es mayor que 300 y hay al menos 2 enrutadores en la red. Cada enrutador "ve" no más de 50 enrutadores.

Ejemplo de entrada

```
6
1-2,3,4
2-1,3
3-1,2,5,6
4-1,5
5-3,4,6
6-3,5
6
1 6
1 5
2 4
2 5
3 6
2 1
10
1-2
2-
3-4
4-8
5-1
6-2
7-3,9
8-10
9-5,6,7
10-8
3
9 10
5 9
9 2
```

Ejemplo de salida

```
-----
1 3 6
1 3 5
2 1 4
2 3 5
3 6
2 1
-----
9 7 3 4 8 10
connection impossible
9 6 2
```

Problema (4) El Subterráneo.

Límite de tiempo: 3 segundos

Te acabas de mudar a una ciudad grande y ruidosa. En lugar de ir en bicicleta a la universidad todos los días, ahora puedes caminar y tomar el metro. Como no quieres llegar tarde a clase, quieres saber cuánto tiempo te llevará llegar a la escuela.

Caminas a una velocidad de 10 km/h. El metro viaja a 40 km/h. Suponga que tiene suerte, y cada vez que llega a una estación de metro, hay un tren que puede abordar de inmediato. Puede subir y bajar del metro varias veces, y puede cambiar entre diferentes líneas de metro si lo desea. Todas las líneas de metro van en ambas direcciones.

Entrada

La entrada comienza con un solo entero positivo en una línea que indica el número de casos siguientes, cada uno de ellos como se describe a continuación. A esta línea le sigue una línea en blanco, y también hay una línea en blanco entre dos entradas consecutivas.

La entrada consiste en las coordenadas x, y de su hogar y la universidad, seguidas de las especificaciones de varias líneas de metro. Cada línea de metro consiste en las coordenadas enteras no negativas x, y de cada parada en la línea, presentadas en orden. Puede suponer que el metro se mueve en línea recta entre las paradas adyacentes, y las coordenadas representan un número entero de metros. Cada línea tiene al menos dos paradas. El final de cada línea de metro es seguido por el par de coordenadas ficticias '-1, -1'. En total hay como máximo 200 paradas de metro en la ciudad.

Salida

Para cada caso de prueba, la salida debe seguir la descripción a continuación. Las salidas de dos casos consecutivos estarán separadas por una línea en blanco.

La salida es el número de minutos que le llevará llegar a la escuela, redondeado al minuto más cercano, tomando la ruta más rápida.

Ejemplo de entrada
1

0 0 10000 1000
0 200 5000 200 7000 200 -1 -1
2000 600 5000 600 10000 600 -1 -1

Ejemplo de salida
21

Problema (5) Colocar a los Guardias.

Límite de tiempo: 3 segundos

En una ciudad del interior hay algunas calles y cruces. Cada calle conecta 2 cruces. El intendende quiere colocar algunos guardias en algunos cruces para que todos los cruces y calles puedan ser custodiados por ellos. Un guardia en un cruce puede proteger todos los cruces y calles adyacentes. Pero los guardias no son gentiles. Si una calle está protegida por múltiples guardias, entonces comienzan a pelear. Por lo tanto, el intendente no quiere el escenario en el que una calle puede estar protegida por dos guardias. Dada la información sobre las calles y cruces de la ciudad, ayuda al intendente a encontrar el número mínimo de guardias necesarios para proteger todos los cruces y calles de su ciudad.

Entrada

La primera línea de la entrada contiene un único número entero $T(T < 80)$ que indica el número de casos de prueba. Cada caso de prueba comienza con 2 enteros v $(1 \leq v \leq 200)$ y e $(0 \leq e \leq 10000)$. v es el número de cruces y e es el número de calles. Cada una de la siguientes e líneas contiene 2 enteros f y t que denotan que hay una calle entre f y t. Todos los cruces están numerados del 0 a $v - 1$.

Salida

Para cada salida de caso de prueba en una sola línea, un número entero m denota el número mínimo de guardias necesarios para proteger todos los cruces y calles. Establezca el valor de m como -1 si es imposible colocar a los guardias sin pelear.

Ejemplo de entrada

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

Ejemplo de salida

```
2
-1
```

Problema (6) Problema de las n Reinas.

Límite de tiempo: 5 segundos

El problema de las n reinas es conocido por todas las personas que han estudiado *backtracking*. En este problema, se debe contar el número de ubicaciones de n reinas en un tablero de $n \times n$ para que no se ataquen dos reinas. Para hacer el problema un poco más difícil (¿más fácil?), hay algunos cuadrados malos donde no se pueden colocar reinas. Tenga en cuenta que los cuadros malos no se pueden usar para bloquear el ataque de las reinas.

Si dos soluciones se vuelven iguales después de algunas rotaciones y reflexiones, se consideran diferentes. Entonces, hay exactamente 92 soluciones para el problema tradicional de las 8 reinas.

Entrada

La entrada consta de un máximo de 10 casos de prueba. Cada caso contiene un número entero n ($3 \leq n \leq 15$) en la primera línea. Las siguientes n líneas representan el tablero, donde los cuadrados vacíos se representan con puntos '.', Los cuadrados defectuosos se representan con asteriscos '*'. El último caso es seguido por un solo cero, que no debe procesarse.

Salida

Para cada caso de prueba, imprima el número de caso y el número de soluciones.

Ejemplo de entrada

```
8
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
4
. * . .
. . . .
. . . .
. . . .
0
```

Ejemplo de salida

```
Case 1: 92
Case 2: 1
```

Anexo 1: Código ejemplo

```
// @JUDGE_ID:  1000AA  100  Java  "Easy algorithm"

import java.io.*;
import java.util.*;

class Main
{
    static String ReadLn (int maxLg)  // utility function to read from stdin
    {
        byte lin[] = new byte [maxLg];
        int lg = 0, car = -1;
        String line = "";

        try
        {
            while (lg < maxLg)
            {
                car = System.in.read();
                if ((car < 0) || (car == '\n')) break;
                lin [lg++] += car;
            }
        }
        catch (IOException e)
        {
            return (null);
        }

        if ((car < 0) && (lg == 0)) return (null);  // eof
        return (new String (lin, 0, lg));
    }

    public static void main (String args[])  // entry point from OS
    {
        Main myWork = new Main();  // create a dinamic instance
        myWork.Begin();            // the true entry point
    }

    void Begin()
    {
        String input;
        StringTokenizer idata;
        int a, b, min, max, num, n, cycle, cyclemax;

        while ((input = Main.ReadLn (255)) != null)
        {
          idata = new StringTokenizer (input);
          a = Integer.parseInt (idata.nextToken());
          b = Integer.parseInt (idata.nextToken());
          if (a < b) { min=a; max=b; } else { min=b; max=a; }
          for (cyclemax=-1, num=min; num<=max; num++) {
            for (n=num, cycle=1; n != 1; cycle++) if ((n % 2) != 0) n=3*n+1; else n >>= 1;
```

```
            if (cycle > cyclemax) cyclemax=cycle;
            }
          System.out.println (a + " " + b + " " + cyclemax);
        }
    }
}
```