ALGORITMIA

Grado en Ingeniería Informática del Software - Curso 2022-2023

GUIÓN DE LA PRÁCTICA 6

OBJETIVO:

Algoritmos vuelta atrás o BACKTRACKING

PRIMERA PARTE: LA HERENCIA MILLONARIA

Acabamos de recibir una herencia de un familiar millonario que se fue a "hacer las Américas" hace bastantes años. El Testamento dice textualmente:

" … y a mi pariente, el informático, le dejo 15 cuentas bancarias. La cantidad de dólares que hay en cada una de ellas corresponde con su clave de acceso y por razones de seguridad, a continuación, únicamente cito el código MD5 asociado a cada una de esas cuentas bancarias:

Ver Fichero "herenciaEnvenenadaMD5.txt"

"

De las claves sí sabemos que tienen una cantidad de caracteres en el intervalo [3 .. 50] y que cada carácter es cualquiera de las 26 letras mayúsculas ["A".."Z"] (sin la "Ñ").

La única forma de cobrar el dinero de cada cuenta es descifrar su clave a partir del código conocido MD5 y para ello **se pide** implementar una Clase RevientaClaves, que se puede ejecutar así:

java paquete.RevientaClaves "herenciaEnvenenadaMD5.txt"

MD5 es un algoritmo criptográfico, que a partir de un texto cualquiera le hace corresponder una secuencia de 128 bits, o lo que es lo mismo, una secuencia de 32 símbolos hexadecimales ["0".."f"].

La ayuda que tenemos es el método JAVA **getMD5** (devuelve el código MD5 de cualquier texto):

```
package alg77777777.p6;
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class EjemploMD5
{
public static String getMD5(String input) {
 try {
 MessageDigest md = MessageDigest.getInstance("MD5");
 byte[] messageDigest = md.digest(input.getBytes());
 BigInteger number = new BigInteger(1, messageDigest);
 String hashtext = number.toString(16);

 while (hashtext.length() < 32) {
 hashtext = "0" + hashtext;
 }
```

```
 return hashtext;
 }
 catch (NoSuchAlgorithmException e) {
 throw new RuntimeException(e);
 }
 }

public static void main (String arg [] )
{
  String clave=arg[0];  //introducir clave posible
  System.out.print(getMD5(clave));

} // fin de main

} // fin de clase
```

Se pide hacer una tabla del tiempo de ejecución de la clase **RevientaClaves**, para cada uno de 15 casos, o sea, para descifrar cada clave y así podamos heredar la cuenta asociada.

SEGUNDA PARTE: EL VIAJANTE DE COMERCIO

El viajante debe partir de una ciudad (es igual una que otra) y visitar todas las demás regresando al final a la ciudad de origen, recorriendo la menor distancia posible (ese ciclo de coste mínimo se llama ciclo de Hamilton o hamiltoniano). En el caso de que dos o más ciclos empaten a un valor mínimo, vale cualquier de ellos.

El problema queda modelizado mediante un grafo completo, dirigido y etiquetado (el peso de cada arista es la distancia positiva >0 entre el par de nodos correspondiente).

Por ejemplo, sea n=4 nodos o ciudades; (v=[0,1,2,3]) con matriz de pesos:

grafo04.txt:

0	10	20	14
11	0	19	12
13	18	0	20
12	14	11	0

Existen **(n-1)!** ciclos diferentes que puede hacer el viajante, en este caso 3!=6 :

Ciclo1) NODO0 > NODO1 > NODO2 > NODO3 > NODO0 = 10+19+20+12 = 61

Ciclo2) NODO0 > NODO1 > NODO3 > NODO2 > NODO0 = 10+12+11+13 = 46

Ciclo3) NODO0 > NODO2 > NODO1 > NODO3 > NODO0 = 20+18+12+12 = 62

Ciclo4) NODO0 > NODO2 > NODO3 > NODO1 > NODO0 = 20+20+14+11 = 65

Ciclo5) NODO0 > NODO3 > NODO1 > NODO2 > NODO0 = 14+14+19+13 = 65

Ciclo6) NODO0 > NODO3 > NODO2 > NODO1 > NODO0 = 14+11+18+11 = 54

Para este fichero grafo04.txt, el ciclo hamiltoniano a seguir por el viajante es:

NODO0 > NODO1 > NODO3 > NODO2 > NODO0

COSTE= 46

Para el caso el grafo contenido en el fichero grafo07.txt, la solución sería:

NODO0 > NODO3 > NODO2 > NODO1 > NODO5 >NODO6 > NODO4 > NODO0

COSTE=141

Para el caso el grafo contenido en el fichero grafo10.txt, la solución sería:

NODO0 > NODO5 > NODO1 > NODO2 > NODO4 > NODO8 > NODO6 > NODO3 > NODO7 > NODO9 > NODO0

COSTE= 262

Se pide:

*Implementar una clase ViajanteBK, que calcule y escriba el ciclo hamiltoniano y su coste mínimo asociado (los datos de entrada estarán en un fichero como los vistos).

java paquete.ViajanteBK nombreFicheroEntrada

*Implementar una clase ViajanteTiemposBK, que vaya simulando el crecimiento del tamaño del problema (para valores n=10, 11, 12, 13, 14, 15, …) y mida tiempos para cada tamaño n. En este caso, los pesos o distancias de cada arista serán generados de forma aleatoria en el rango [10..99]. Con los tiempos obtenidos, ha de realizar la tabla pertinente.

-

Se entregará:

- *Los ficheros fuente de las clases, que se hayan programado, dentro del paquete o del proyecto Eclipse.*
- *Clases de prueba*
- *Un documento PDF con la tabla de tiempos cubierta y la explicación de la correspondencia con la complejidad teórica*

*Se habilitará una tarea en el campus virtual para realizar la entrega. Esta práctica se trabajará durante una sesión y el **plazo límite es un día antes** de la próxima sesión de prácticas de tu grupo.*