PRACTICAS TABLERO - SESION 5

EJERCICIO 1

Resolver el Sudoku (estudiar y ejecutar las clases *s5.SudokuUna.java* y *s5.SudokuTodas.java*).

EJERCICIO 2

Calcular el ciclo de Hamilton en un grafo completo no dirigido (o simétrico) con pesos de aristas de valor positivo. Este problema se llama también **problema del viajante** y hay que calcular el ciclo que pase por todos los nodos del grafo de coste mínimo.

Se pide:

a) mediante la técnica "devoradora".

b) mediante la técnica "vuelta atrás".

SOLUCIÓN

a) el problema a resolver es el **CICLO ÓPTIMO DEL VIAJANTE (CICLO DE HAMILTON)** en un grafo completo no dirigido de n nodos.

Los algoritmos que desarrollamos son estos dos devoradores:

1)devorador1 = a partir del primer nodo (v[0]) ir visitando el nodo aún no visitado que esté más próximo, hasta visitarlos todos. Desde el último se regresará a casa (v[0]).

2)devorador2 = aplicar n veces el algoritmo anterior **devorador1** (cada vez un nodo inicial diferente), para calcular de los n ciclos hallados, el de menor coste.

La clase adjunta *s5.ViajanteDevorador* implementa ambos heurísticos. Como grafo ejemplo se toma el siguiente:

Pesos	0	1	2	3	4	5	6
0	inf.	12	43	99	57	32	78
1	12	inf.	10	80	93	33	11
2	43	10	inf.	60	43	20	22
3	99	80	60	inf.	50	18	31
4	57	93	43	50	inf.	31	73
5	32	33	20	18	31	inf.	22
6	78	11	22	31	73	22	inf.

```
n=7;
w[0][1]=12;w[1][0]=12;w[0][2]=43;w[2][0]=43;
w[0][3]=99;w[3][0]=99;w[0][4]=57;w[4][0]=57;
w[0][5]=32;w[5][0]=32;w[0][6]=78;w[6][0]=78;
w[1][2]=10;w[2][1]=10;w[1][3]=80;w[3][1]=80;
w[1][4]=93;w[4][1]=93;w[1][5]=33;w[5][1]=33;
w[1][6]=11;w[6][1]=11;w[2][3]=60;w[3][2]=60;
w[2][4]=43;w[4][2]=43;w[2][5]=20;w[5][2]=20;
w[2][6]=22;w[6][2]=22;w[3][4]=50;w[4][3]=50;
w[3][5]=18;w[5][3]=18;w[3][6]=31;w[6][3]=31;
```

```
w[4][5]=31;w[5][4]=31;w[4][6]=73;w[6][4]=73;
w[5][6]=22;w[6][5]=22;
for (int i=0;i<w.length;i++)
   w[i][i]=Integer.MAX_VALUE;
```

RESULTADO de *java s5.ViajanteDevorador* **:**

EL COSTE DEL CICLO DEVORADOR1 ES= 221
EL CICLO ES=
NODO0—NODO1—NODO2—NODO5—NODO3—NODO6—NODO4—NODO0

EL COSTE DEL CICLO DEVORADOR2 ES= 201
EL CICLO ES=
NODO4—NODO5—NODO3—NODO6—NODO1—NODO2—NODO0—NODO4

Se comprueba que ninguno de los dos calcula la solución óptima, ya que en este caso el ciclo de Hamilton es:

EL COSTE DEL CICLO DE HAMILTON ES= 181
EL CICLO ES=
NODO0—NODO1—NODO2—NODO6—NODO3—NODO5—NODO4—NODO0

Asimismo se adjunta *s5.ViajanteDevoradorTiempos*, que simula el crecimiento del problema (n = número de nodos el grafo) y mide tiempos para ambos devoradores.

Haciendo *java s5.ViajanteDevoradorTiempos* se obtienen los siguientes tiempos en un determinado ordenador:

n	*tiempo devorador1 (microsg.)*
400	*t1*
800	*4t1*
1600	*16t1*

*(concuerda con el n**2 analizado).*

N	*tiempo devorador2 (milisg.)*
400	*t2*
800	*8t2*
1600	*64t2*

*(concuerda con el n**3 analizado).*

Este problema del viajante es un problema NP, lo que supone que no hay ningún algoritmo polinómico capaz de darle solución. Dicho lo anterior, abordamos el problema por "backtracking".

De ello se encargan las siguientes 4 clases adjuntas:

1)s5.ViajanteTodos

calcula todos los ciclos que puede seguir el viajante. El número de ciclos es (n-1)!. Como el grafo que se toma como ejemplo es el mismo de antes (n=7), devuelve 6!=720 ciclos.

java alg77777777.s6.ViajanteTodos

2)s5.ViajanteMejor

similar a la anterior, pero una vez calculados todos nos dice el mejor mediante su simple comparación.

java alg77777777.s6.ViajanteMejor

3)s5.ViajanteMejorPoda

incorpora poda cuando el coste hasta un nodo dado ya es mayor o igual que la mejor solución encontrada hasta entonces. Eso se puede hacer porque se está calculando un mínimo y los pesos de las aristas son positivos.

java alg77777777.s6.ViajanteMejorPoda

4)s5.ViajanteMejorTiempos

recibe como argumento el tamaño n del grafo y calcula el tiempo que tardan los dos métodos anteriores (sin poda y con poda) para ese n.

java s5.ViajanteMejorTiempos n // número de nodos como argumento