Algoritmos http://www.lfcia.org/alg/ 2007-2008

Ingeniería Informática

Ingeniería Técnica de Informática de Gestión

Práctica 5

Fecha límite de entrega: martes, 15 de enero

Grafos

Problema del camino mínimo

A continuación se presenta un pseudocódigo para calcular el camino mínimo de cada vértice a los restantes en grafos ponderados siguiendo el algoritmo de *Dijkstra*. El argumento es la matriz de adyacencia del grafo y el resultado es una tabla con las distancias mínimas desde cada vértice a los restantes.

```
función CaminosMínimos( M[1..n,1..n] ): matriz
  para m := 1 hasta n hacer
    noVisitados := { 1, 2, ..., m-1, m+1, ..., n };
    para i := 1 hasta n hacer
      Distancias[m, i] := M[m, i]
    fin para
    repetir n-2 veces:
      v := nodo de noVisitados que minimiza Distancias[m, v];
      noVisitados := noVisitados - { v };
      para cada w en noVisitados hacer
        si Distancias[m, w] > Distancias[m, v] + M[v, w]
        entonces Distancias[m, w] := Distancias[m, v] + M[v, w]
        fin si
      fin para
    fin repetir
  fin para
  devolver Distancias
fin función
```

Se pide:

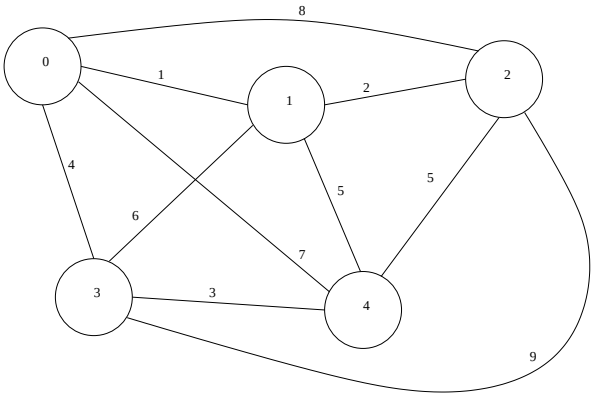
1. Implemente en JAVA el algoritmo presentado (figura 1).
2. Valide el correcto funcionamiento de la implementación. En las figuras 2 y 3 se proponen dos casos de prueba.
3. Usando las clases de las figuras 4 y 5 para generar aleatoriamente grafos completos no dirigidos, calcule empíricamente la complejidad computacional del algoritmo para el cálculo de las distancias mínimas.

```
class CaminosMinimos {
    //     Camino mínimo de cada vértice a los restantes. Se utiliza el
    //     algoritmo de Dijkstra.
    public static int [][] caminosMinimos(int [][] m) {
        int[][] distancias = new int[m.length][m.length];
    /*
      ...
    */
        return distancias;
    }/* -- fin del algoritmo -- */

//...

}/* -- fin de la clase -- */
```

Figura 1: La clase `CaminosMinimos`



(a) Grafo

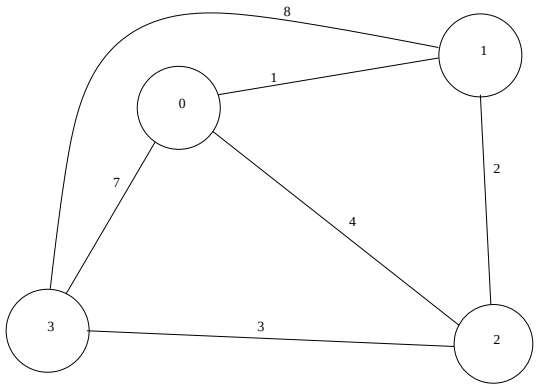
$$\begin{pmatrix} 0 & 1 & 8 & 4 & 7 \\ 1 & 0 & 2 & 6 & 5 \\ 8 & 2 & 0 & 9 & 5 \\ 4 & 6 & 9 & 0 & 3 \\ 7 & 5 & 5 & 3 & 0 \end{pmatrix}$$

(b) Matriz de adyacencia

	0	1	2	3	4
0	0	1	3	4	6
1	1	0	2	5	5
2	3	2	0	7	5
3	4	5	7	0	3
4	6	5	5	3	0

(c) Distancias mínimas

Figura 2: Primer ejemplo



(a) Grafo

$$\begin{pmatrix} 0 & 1 & 4 & 7 \\ 1 & 0 & 2 & 8 \\ 4 & 2 & 0 & 3 \\ 7 & 8 & 3 & 0 \end{pmatrix}$$

(b) Matriz de adyacencia

	0	1	2	3
0	0	1	3	6
1	1	0	2	5
2	3	2	0	3
3	6	5	3	0

(c) Distancias mínimas

Figura 3: Segundo ejemplo

```
public interface InicializacionMatrices {
    void inicializar(int [][] m);
}
```

Figura 4: La interfaz `InicializacionMatrices`

```
public class MatrizAleatoria implements InicializacionMatrices {
    java.util.Random rand = new java.util.Random();
    public void inicializar(int [][] m) {
        for (int i=0; i<m.length; i++)
            for (int j=i+1; j<m.length; j++)
                m[i][j] = Math.abs(rand.nextInt()) % 1001;
        for (int i=0; i<m.length; i++)
            for (int j=0; j<=i; j++)
                if (i==j)
                    m[i][j] = 0;
                else
                    m[i][j] = m[j][i];
    }
    public String toString() {
        return "Inicializacion aleatoria [1..1000] de una grafo completo" +
            "no dirigido, representado por una matriz de adayencia";
    }
}
```

Figura 5: La clase `MatrizAleatoria`