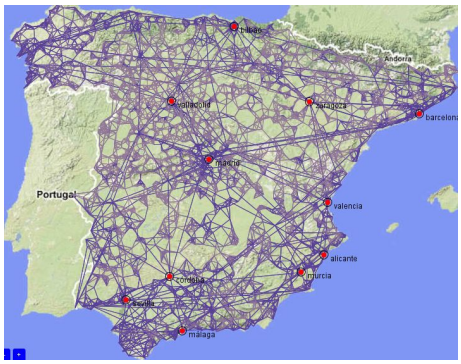
EDA (E.T.S. de Ingeniería Informática)- Curso 2015-2016

Práctica 6. Municipios de una red como aplicación de la exploración de un *Grafo* (3 sesiones)

Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València

1. Introducción y objetivos de la práctica

Son muchos los problemas prácticos de interés que pueden ser planteados en términos de problemas sobre grafos. El objetivo de esta práctica es la implementación y utilización del algoritmo de **Dijkstra** para calcular el camino más corto desde un cierto vértice del *Grafo* hasta el resto de vértices. En concreto, el *Grafo* con el que se va a trabajar representa un conjunto de municipios del país y las distancias que los separan punto a punto medidas en kilómetros. A pesar del elevado número de municipios y de carreteras, el algoritmo de *Dijkstra* permite encontrar el camino más corto entre dos municipios de forma muy eficiente.



2. Planteamiento y actividades

El algoritmo de *Dijkstra*, también llamado de *caminos mínimos*, es un algoritmo para la determinación del camino más corto desde un vértice origen al resto de vértices en un grafo dirigido, con pesos en cada arista y sin ciclos de peso negativo. Su nombre se refiere a *Edsger Dijkstra*, quien lo describió por primera vez en 1959. La labor del alumno consiste en implementar de forma eficiente este algoritmo y un método que, a partir de la información resultante de este algoritmo, obtenga la secuencia de vértices que conforman el camino más corto hasta un vértice destino dado. Como aplicación de grafos se planteará la respresentación de una red de municipios conectados por carretera y como aplicación de *Dijkstra* la obtención del camino más corto entre dos municipios dados.

Para construir una aplicación que permita consultar la ruta más corta entre dos municipios cualesquiera se deberán realizar las siguientes actividades:

2.1. Actividad #1: Creación del paquete *librerias.estructurasDeDatos.grafos*

El alumno deberá crear el nuevo paquete *librerias.estructurasDeDatos.grafos*, que debe contener la jerarquía de clases definida para la implementación de un grafo dirigido y con pesos. Este paquete deberá incluir las siguientes clases (que se encuentran disponibles en *PoliformaT*):

- `Grafo`: define la funcionalidad básica de un grafo, abstrayéndose de los detalles de implementación. Algunos métodos, como los recorridos y las búsquedas de caminos mínimos, se implementan en esta clase pues no dependen de las decisiones de implementación del grafo.
- `Adyacente`: esta clase contiene la información necesaria para representar los adyacentes a un vértice del grafo en su implementación mediante listas de adyacencia. Así, un *Adyacente* tiene un destino, que es el código del vértice adyacente, y un peso, que es el peso o coste de la arista que une ambos vértices.
- `GrafoDirigido`: implementación de un grafo dirigido con pesos mediante listas de adyacencia. Hereda la funcionalidad básica de la clase `Grafo`.

2.2. Actividad #2: implementación del algoritmo de *Dijkstra* y decodificación del camino más corto

El alumno deberá completar los siguientes métodos de la clase `Grafo`:

- El método `dijkstra`: recibe como parámetro *origen*, que es el código del vértice de partida y debe construir los dos arrays siguientes:
  - `protected double[] distanciaMin;` // Distancia mínima del origen al resto de vértices
  - `protected int[] caminoMin;` // Vértice anterior en el camino más corto

  Un esquema de una implementación eficiente del algoritmo es el siguiente:

```
inicializar los arrays distanciaMin, caminoMin y visitados
inicializar la cola de prioridad cp

while (no vacía cp) {
    Seleccionar el vértice v, mínimo de cp;
    if (visitados[v] == 0) {
        visitados[v] = 1;
        Para todo w en adyacentesDe(v) {
            if (distanciaMin[w] > distanciaMin[v] + coste de v a w) {
                  distanciaMin[w] = distanciaMin[v] + coste de v a w;
                  caminoMin[w] = v;
                  insertar en cp el par (w, distanciaMin[w]);
            }
        }
    }
}
```

  La interfaz `ColaPrioridad` se encuentra en el paquete *librerias.estructurasDeDatos.modelos* y una implementación eficiente de la misma es la clase `PriorityQColaPrioridad` que se encuentra ubicada desde la práctica 1 en *librerias.estructurasDeDatos.jerarquicos*. Nótese que los elementos de la Cola de Prioridad que usa `dijkstra` son pares vértice alcanzado y coste hasta ese vértice.

Será necesario definir una clase también en el paquete *librerias.estructurasDeDatos.grafos* para representar estos datos.

- El método `caminoMinimo`: recibe dos parámetros: *origen*, que es el código del vértice de partida (municipio origen), y *destino*, que es el código del vértice destino. Como resultado, el método debe devolver una *Lista con Punto de Interés* con los códigos de los vértices que conforman el camino mínimo entre *origen* y *destino*. Para ello se sugiere seguir los siguientes pasos:

  1. Invocar al algoritmo de *Dijkstra*.
  2. Recuperar el camino mínimo a partir de los arrays *distanciaMin* y *caminoMin*, guardando el camino resultante en una *Lista con Punto de Interés*.

  En caso de que *origen* o *destino* no estén en el grafo, o que no exista camino entre ambos municipios, el método deberá devolver una lista vacía.

Se dispone de los ficheros `TestGrafos1.class` y `TestGrafos2.class` para comprobar el funcionamiento correcto de esta actividad. Deberán copiarse en el paquete *librerias.estructurasDeDatos.grafos*.

2.3. Actividad #3: el paquete *aplicaciones.municipios*

Se deberá crear un nuevo paquete, denominado *aplicaciones.municipios*, que contendrá todas las clases relacionadas con la aplicación. Dentro de este paquete se deberá añadir las siguientes clases disponibles en PoliformaT:

- La clase completa `Municipio` que permite guardar la información relativa a un municipio (nombre, población, extensión y posición dentro del mapa).
- La clase `GestorGrafoMunicipios` dispone internamente de un grafo de municipios y se encarga de gestionar las operaciones habituales del mismo, así como su inicialización a partir de los ficheros de datos. En esta clase se deberá completar el método *caminoMinimo* para que traduzca la lista de códigos obtenida a una lista de etiquetas. Puesto que cada vértice del grafo corresponde a un municipio, es conveniente poder obtener el camino mínimo como una lista de los municipios que lo componen en lugar de como una lista de códigos.

2.4. Actividad #4: aplicación de prueba

Añade la aplicación de prueba, `GuiGrafo`, al paquete *aplicaciones.municipios*. Para poder ejecutarla es necesario copiar los ficheros *municipios.txt*, *distancias.txt* y *spain.jpg*, disponibles en *PoliformaT*, a la carpeta `$HOME/eda/aplicaciones/municipios`.

Al iniciar la aplicación gráfica se puede observar el grafo de municipios con todas sus aristas y alguno de sus vértices más importantes. El grafo consta de un total de 4016 vértices y 47962 aristas. A pesar de su considerable tamaño, se puede comprobar la eficiencia del algoritmo de *Dijkstra*, que es capaz de calcular el camino mínimo entre un municipio y todos los demás de forma casi instantánea.

El resultado que se debe obtener si se consulta a través de la aplicación `GuiGrafo` el camino más corto para llegar de Albaida a Valencia, es el que se muestra en la siguiente figura: