EDA (ETS de Ingeniería Informática). Curso 2019-2020
Practica 6: Una aplicación del algoritmo de Kruskal a la vida real: diseño del tendido eléctrico entre ciudades
Sesión 1: ¿Cómo conectar N ciudades tendiendo (solo) N - 1 cables eléctricos?

Departamento de Sistemas Informáticos y Computación. Universitat Politècnica de València

1. Objetivos

Esta práctica persigue un doble objetivo. Por un lado, al concluir las dos sesiones que la componen, el alumno deberá ser capaz de reconocer y aplicar los conceptos y resultados más relevantes de la teoría de Grafos a la resolución de un tipo de problemas que, bajo múltiples instancias, se plantea en la vida real: conectar entre sí un conjunto de objetos de la forma más "económica" posible, o problema del Árbol de Recubrimiento Mínimo (*Minimum Spanning Tree* o *MST problem* en inglés).

Por otro lado, además, el alumno deberá ser capaz de representar y obtener soluciones factibles y óptimas a este tipo de problemas reutilizando la jerarquía Java `Grafo` que se le proporciona. Específicamente, en esta práctica implementará una versión eficiente del algortimo de Kruskal que permite optimizar el diseño del tendido eléctrico entre un conjunto de ciudades dado, el mismo problema práctico que, en 1926, llevó al matemático checo Otakar Boruvka a formular por primera vez el problema del Árbol de Recubrimiento Mínimo y a resolverlo algorítmicamente.

2. Descripción del problema

Una compañía eléctrica necesita renovar el tendido eléctrico existente entre un grupo de `N` ciudades con, obviamente, el menor coste posible. Como es muy probable que en el tendido "sobren" bastantes cables (ver Figura 1), la compañia decide analizar si es posible redefinir la topología de esta red eléctrica para que solo se mantengan activos y renueven `N-1` de sus cables, que es el número mínimo que debe tener para dar servicio a `N` ciudades. Nótese que esta es solo una **solución factible** al problema, pues minimiza el número de cables a renovar pero no el coste de su renovación; como se verá en la segunda sesión de esta práctica, para decidir cuáles de los cables de la red son los `N-1` a renovar con coste mínimo hay que hacer unos cuantos cálculos más.

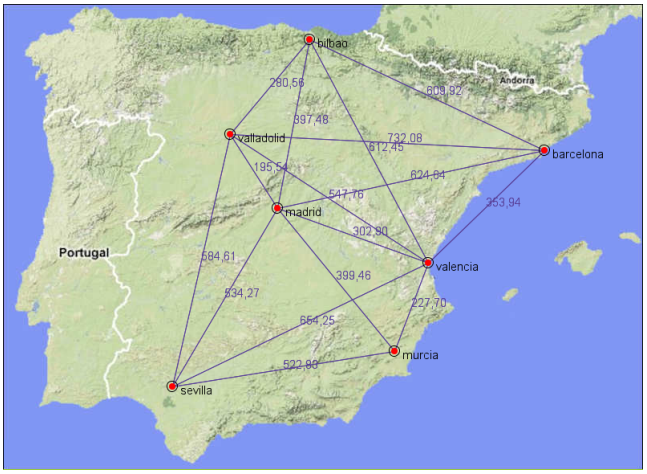

Figura 1: Tendido eléctrico de la red que da servicio a siete ciudades españolas

Para llevar a cabo este estudio de viabilidad, el equipo informático de la compañía representa la red eléctrica a renovar mediante un grafo No Dirigido en el que los vértices son las ciudades de la red y las aristas los cables del tendido eléctrico, etiquetadas con los millones de euros que costaría su renovación. Hecho esto, plantea en términos de Teoría de Grafos si el problema tiene solución para cualquier conjunto de `N` ciudades. El resultado conocido que aplica es que SII el grafo que representa al problema es Conexo, existe al menos un conjunto de `N-1` de sus aristas que permiten conectar entre sí cualquier par de vértices del grafo. Nótese que, al ser `N-1` el

número mínimo de aristas que tiene un grafo No Dirigido y Conexo de `N` vértices, este conjunto de aristas no contiene ninguna arista Hacia Atrás (que forme ciclo) y, además, garantiza que cualquier par de vértices del grafo están conectados por un único camino simple; precisamente por ello, define lo que se denomina un Árbol de Recubrimiento del grafo, i.e. un subgrafo Acíclico (Árbol), Conexo y cuyos `N` vértices son los vértices del grafo.

En base a este resultado, el equipo informático ya sabe a qué problema se enfrenta y, además, qué algoritmo emplear para resolverlo eficientemente: encontrar un Árbol de Recubrimiento de un grafo No Dirigido de `N` vértices, devolviendo como resultado el conjunto de `N-1` aristas que lo componen; si dicho conjunto no existe, porque el grafo no es Conexo, el resultado a devolver es un conjunto inexistente de aristas (o `null`).

Para obtener la solución de este problema, basta con realizar un Recorrido en Anchura (BFS) de un solo vértice `v` del grafo, cualquiera de ellos, y guardar como resultado las aristas empleadas para visitar todos los vértices alcanzables desde `v`. Hecho esto, si el BFS de `v` devuelve un conjunto de exactamente `N-1` aristas, el grafo es Conexo y, por tanto, la solución del problema es la del BFS de `v`; sino, el grafo no es Conexo y la solución del problema es `null`. Para ilustrar este resultado, en la siguiente figura se muestran con líneas gruesas negras los Árboles de Recubrimiento resultado de los BFS de los vértices Barcelona (Figura 2(a)) y Sevilla (Figura 2(b)) del grafo ejemplo (Conexo) de la Figura 1.

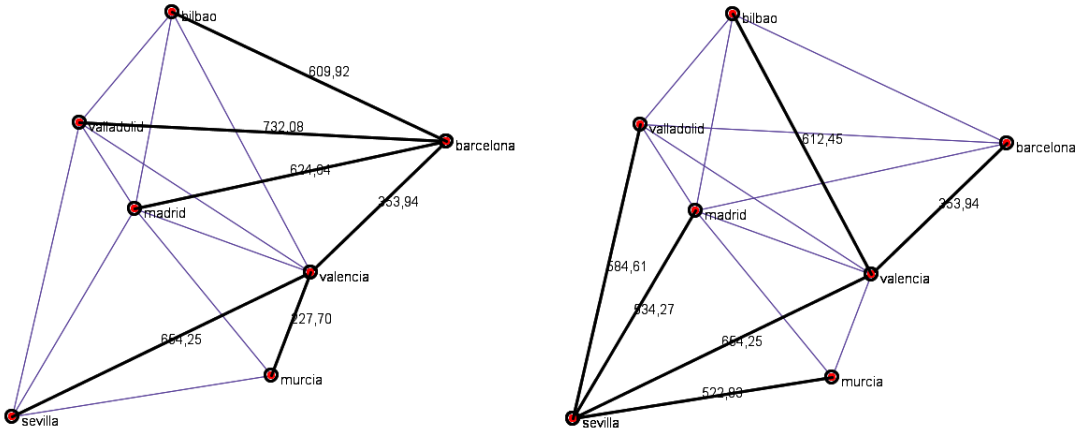

Figura 2: Dos Árboles de Recubrimiento BFS del grafo de la Figura 1

Ya para concluir, es importante recordar que cualquier Árbol de Recubrimento que se obtenga al realizar el BFS de un vértice de un grafo es solo una solución factible al problema planteado inicialmente, i.e. no es necesariamente un Árbol de Recubrimiento Mínimo; de hecho, solo lo sería si todas las aristas del grafo tuvieran el mismo peso, lo que es altamente improbable en la vida real... Justamente por ello, ninguno de los dos Árboles de Recubrimiento de la Figura 2 es un Árbol de Recubrimiento Mínimo del grafo de la Figura 1. Como se detallará en la segunda sesión de esta práctica, para obtener un Árbol de Recubrimiento Mínimo de un grafo (ver Figura 3) habrá que modificar la estrategia BFS empleada hasta el momento para optimizar la selección de las aristas del Árbol de Recubrimiento "a la Kruskal", es decir, en base a sus pesos.

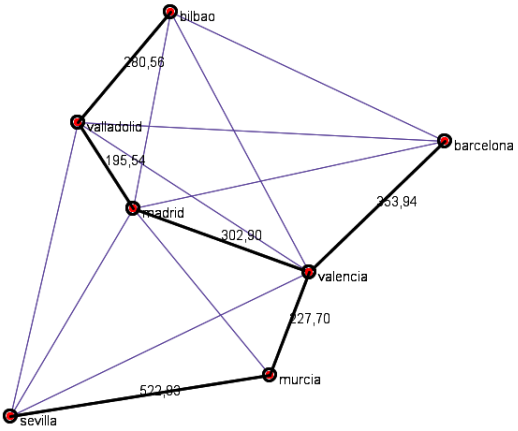

Figura 3: Árbol de Recubrimiento Mínimo del grafo de la Figura 1

3. Actividades

Antes de realizar las actividades que se proponen en esta sesión, el alumno deberá actualizar la estructura de paquetes y clases de su proyecto *BlueJ eda* tal y como se indica a continuación:

- Abrir su proyecto *BlueJ eda* y, dentro de su paquete *librerias.estructurasDeDatos*, crear un nuevo paquete de nombre *grafos*. Hecho esto, ***Salir*** de *BlueJ*.
- Descargar en su carpeta *grafos* (correspondiente al paquete del mismo nombre que acaba de crear en *BlueJ*) todos los ficheros disponibles en *PoliformaT* para esta primera sesión de la práctica.
- Invocar de nuevo a *BlueJ* y acceder al paquete *librerias.estructurasDeDatos.grafos* de su proyecto *eda*. Tal como muestra la siguiente figura (Figura 4), este paquete debe contener ahora las clases de la Jerarquía Java `Grafo` y dos clases más: `Arista` y `TestGrafo`. Una vez completada, la primera se incorporará a la jerarquía `Grafo`, pues la usan los nuevos métodos que el alumno debe implementar en esta práctica; la segunda servirá, como su nombre indica, para comprobar la corrección del código desarrollado durante esta sesión.

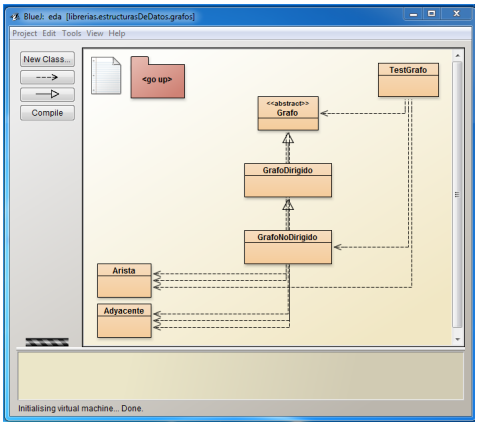

Figura 4: Clases del paquete *grafos*

3.1. Implementar el método `arbolRecubrimientoBFS` y completar la clase `Arista`

En esta actividad el alumno debe implementar en la clase `Grafo` un nuevo método de Búsqueda BFS que devuelva una solución factible al problema del Árbol de Recubrimiento Mínimo. Su especificación y cabecera ya figuran en la clase `Grafo` y son como sigue:

```
/** PRECONDICIÓN: !this.esDirigido()
 *  Devuelve un subconjunto de aristas que conectan todos los vertices
 *  de un grafo No Diridigo y Conexo, o null si el grafo no es Conexo.
 *  @return Arista[], array con las numV - 1 aristas que conectan los numV
 *                    vertices del grafo, o null si el grafo no es Conexo
 */
public Arista[] arbolRecubrimientoBFS()
```

A partir de ellas, resulta obvio que antes de implementar `arbolRecubrimientoBFS` es necesario completar el código de la clase `Arista`; para ello, basta seguir los comentarios que aparecen en su código.

En cuanto a la implementación de `arbolRecubrimientoBFS`, se recomienda no empezarla desde cero, sino como sigue:

- Copiar el cuerpo del método `toArrayBFS()` que hay en la clase `Grafo` en el de `arbolRecubrimientoBFS`.
- Modificar y/o eliminar aquellas líneas del cuerpo actual de `arbolRecubrimientoBFS` en base a la descripción del problema realizada en el apartado 2 de este boletín.
- Copiar el método `protected` que implementa el Recorrido BFS de un vértice dado de un Grafo y modificar tanto su cabecera como su cuerpo para que puede ser invocado desde el cuerpo de `arbolRecubrimientoBFS`.

3.2. Validar el código desarrollado en la práctica

Para comprobar la corrección del código implementado durante la sesión el alumno debe ejecutar el programa `TestGrafo`.