Práctica Individual 2

Problema Número 2

Análisis y Diseño de Datos y Algoritmos / Estructura de Datos y Algoritmos

Grado de Software

Curso 2016-17

Luis Candelario Luna (luisitiluna@hotmail.com)

Tutor: Miguel Toro Bonilla

Índice

1 Enunciado y Cuestiones a resolver

Nos proporcionan un mapa de navegación modelado como un grafo no dirigido compuesto por puertos y trayectos. Los trayectos representan los desplazamientos marítimos navegables de forma directa entre dos puertos sin hacer escala en un puerto intermedio. Los objetos Puerto tendrán las siguientes propiedades:

• id: String **// Identificador del puerto.**
• pais: String **// País donde se encuentra el puerto.**
• tasas: Double **// Tasas que hay que pagar por atracar en el puerto (en miles de euros).**
• minutosHastaElAeropuerto: Integer **// Minutos necesarios hasta llegar al aeropuerto más cercano.**
Negativo si no hay aeropuertos cercanos.
• atractivo: Integer **// Atractivo turístico del enclave. El atractivo podrá tomar valores entre 1 y 3.**
1 para los puertos con menor atractivo y 3 para los puertos con mayor atractivo.
• calado: Integer **// Calado del puerto (en metros).**
• esPuntoDeAbastecimiento: Boolean **// Indica si el puerto permite abastecer los cruceros.** Y los objetos Trayecto tendrán las siguientes propiedades:
• origen: Puerto **// Uno de los puertos que une el trayecto.**
• destino: Puerto **// El otro de los puertos que une el trayecto.** • costeTrayecto: Double **// Coste del trayecto entre puertos (en miles de euros).**
• duracionEnDias: Integer **// Duración del trayecto (en días).**
• nivelDeSeguridad: Integer **// Nivel de seguridad del trayecto, podrá tomar valores entre 1 y 3. 1 para los trayectos con menor seguridad y 3 para los trayectos con mayor seguridad.**

También se representará las travesías de los cruceros mediante listas de puertos, List<Puerto>. La travesía de un crucero representa la secuencia de escalas que realiza el crucero durante su viaje, cada par de escalas representan un trayecto directo entre dos puertos. A partir de la información disponible en el grafo de navegación nos piden completar el código de la clase *MapaDeNavegacion* y el resto de clases necesarias:

```
public class MapaDeNavegacion {
private MapaDeNavegacion() { }
private static Graph<Puerto,Trayecto> g = null;
1) Lee de un fichero el mapa de navegación y lo almacena en el atributo g. Utilice la clase GraphsReader e
implemente los métodos create de las clases Puerto y Trayecto.
public static MapaDeNavegacion read (String file) { // TODO }
2) Devuelve el mapa de navegación.
public Graph<Puerto,Trayecto> getMap(){ // TODO }
3) Devuelve el puerto asociado al identificador id.
public Puerto getPuertoById(String id) { // TODO }
4) Devuelve una lista con el menor número de puertos necesarios para controlar los desplazamientos de todos los
buques.
public Set<Puerto> getRedDeControles() { // TODO }
5) Devuelve las zonas de influencia de la red de puertos condicionada por el atributo limiteDeDiasXDesplazamiento.
Se define una zona de influencia como todos los puertos alcanzables entre ellos mismos utilizando sólo
desplazamientos con una duración menor o igual que el limiteDeDiasXDesplazamiento. Las zonas de influencias
deberán tener al menos tres puertos.
public List<Set<Puerto>> getZonasDeInfluencia(int limiteDeDiasXDesplazamiento) { // TODO
}
6) Devuelve el conjunto mínimo de trayectos necesarios para acceder a todos los puertos del mapa de navegación
con una red de ferris. Los ferris son naves que solo realizan desplazamientos directos entre puertos.
public Set<Trayecto> getRedDeFerris() { // TODO }
7) Devuelve la duración en días de la travesía de un crucero.
public Integer getDuracionDeCrucero(List<Puerto> travesia) { // TODO }
8) Devuelve el atractivo medio de la travesía de un crucero.
public Double getAtractivoMedio(List<Puerto> travesia) { // TODO }
9) Devuelve un booleano indicando si existe algún punto de abastecimiento durante la travesía de un crucero.
public boolean existeAlgunPuntoAbastecimiento(List<Puerto> travesia) { // TODO }
10) Devuelve el coste de la travesía de un crucero, incluye las tasas de los puertos y los costes de los desplazamientos
de los trayectos.
public Integer getCosteXCrucero(List<Puerto> travesia) { // TODO }
11) Devuelve la lista de puertos con aeropuertos cercanos. Para que un aeropuerto sea cercano debe estar a menos de
61 minutos de distancia del puerto.
public Set<Puerto> getPuertosConAeropuertosCercanos() { // TODO }
12) Devuelve un conjunto con todas las travesías de cruceros válidas. Una travesía es válida si es un camino mínimo
entre dos puertos que tengan cada uno un aeropuerto cercano. Si existe más de un
```

camino mínimo con estas características se podrá escoger cualquiera de ellos.
public Set<List<Puerto>> getCrucerosValidos() { **// TODO** }
13) Devuelve un conjunto con todos los cruceros fantásticos. Un crucero fantástico será un crucero válido que tiene que durar como mínimo 9 días y como máximo 12. Además tiene que tener un atractivo turístico mayor o igual a 2.3
y poseer algún puerto donde abastecerse.
public Set<List<Puerto>> getCrucerosFantasticos() { **// TODO** }
}

2.- Seguimiento

2.1 Control de seguimiento y tutorías

El alumno debe listar y detallar las visitas a tutoría

Fecha	Profesor	Detalles y dudas resueltas
13/12/2016	Luisa Parody	Problemas de comprensión de enunciado resueltos

2.2 Portafolio

11/12/2016 Realización de los métodos más básicos del ejercicio y planteamiento de los métodos con más dificultad para preguntarlos durante el seguimiento.

20/12/2016 Implementación de las clases Puerto y Trayecto a través del uso de factorías y comprobación de los resultados obtenidos al ejecutar los métodos realizados.

27/12/2016 Realización de parte de los métodos con más dificultad comprobando sus resultados al ejecutarlos.

05/01/2017 Implementación de los métodos restantes y su respectiva eficiencia.

08/01/2017 Realización de la plantilla de la memoria para proceder a la entrega.

El ejercicio se ha realizado directamente sobre Eclipse sin ayuda de ningún material impreso realizado a mano, únicamente a base de test y de conocimientos adquiridos en la asignatura Matemática Discreta.

3.-Implementación

Tengo problemas a la hora de copiar el código y no consigo que copie el formato original.

Clase Puerto

```
package us.lsi.graphs.cruceros;

import us.lsi.graphs.StringVertexFactory;

public class Puerto {

	static public StringVertexFactory<Puerto> factoria = new
StringVertexFactory<Puerto>() {


	public Puerto createVertex(String[] formato) {

		return new
Puerto(formato[0],formato[1],Double.valueOf(formato[2]),Integer.parseI
nt(formato[3]),Integer.parseInt(formato[4]),

	Integer.parseInt(formato[5]),Boolean.valueOf(formato[6]));
	}
	};

	private static  Double doubleValueOf(String[] formato, int pos)
{
		if (pos<formato.length) {
			return Double.valueOf(formato[pos]);
		}
		return null;
	}

	private static  Integer intValueOf(String[] formato, int pos) {
		if (pos<formato.length) {
			return Integer.valueOf(formato[pos]);
		}
		return null;
	}
```

```
	private static  Boolean boolValueOf(String[] formato, int pos) {
		if (pos<formato.length) {
			return Boolean.valueOf(formato[pos]);
		}
		return null;
	}


	private String id = null;
	private String pais = null;
	private Double tasas = null;
	private Integer minutosHastaElAeropuerto = null;
	private Integer atractivo = null;
	private Integer calado = null;
	private Boolean esPuntoDeAbastecimiento = null;

	public Puerto(Puerto p) {
		this(p.id, p.pais,p.tasas, p.minutosHastaElAeropuerto,
p.atractivo,p.calado, p.esPuntoDeAbastecimiento);
	}

	public Puerto(String id, String pais, String[] formato) {
		this(formato[0],
			formato[1],
			doubleValueOf(formato,2),
			intValueOf(formato,3),
			intValueOf(formato,4),
			intValueOf(formato,5),
			boolValueOf(formato,6)
		);
		if (formato.length<6) { System.out.println(this + "->" +
"Numero de campos erroneos"); }
		if (this.getTasas()>40) { System.out.println(this + "->" +
"ERR[2]: Tasas. " + this.getTasas()); }
		if (this.getMinutosHastaElAeropuerto()>100) {
System.out.println(this + "->" + "ERR[3]: Minutos hasta el aero. " +
this.getMinutosHastaElAeropuerto()); }
		if (this.getAtractivo()>4) { System.out.println(this + "-
>" + "ERR[4]: Atractivo. " + this.getAtractivo()); }
	  if (this.getCalado()>30) { System.out.println(this + "->" +
"ERR[5]: Calado. " + this.getCalado()); }
	}

	public Puerto(String id, String pais, Double tasas, Integer
minutosHastaElAeropuerto,
			Integer atractivo, Integer calado, Boolean
esPuntoDeAbastecimiento) {
		super();
		this.id = id;
		this.pais = pais;
		this.tasas = tasas;
		this.minutosHastaElAeropuerto = minutosHastaElAeropuerto;
		this.atractivo = atractivo;
		this.calado = calado;
		this.esPuntoDeAbastecimiento = esPuntoDeAbastecimiento;
	}
```

```
	public String getId() {
		return id;
	}

	public String getPais() {
		return pais;
	}

	public Double getTasas() {
		return tasas;
	}

	public Integer getMinutosHastaElAeropuerto() {
		return minutosHastaElAeropuerto;
	}

	public Integer getAtractivo() {
		return atractivo;
	}

	public Integer getCalado() {
		return calado;
	}

	public Boolean esPuntoDeAbastecimiento() {
		return esPuntoDeAbastecimiento;
	}

	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + ((id == null) ? 0 :
id.hashCode());
		return result;
	}

	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		Puerto other = (Puerto) obj;
		if (id == null) {
			if (other.id != null)
				return false;
		} else if (!id.equals(other.id))
			return false;
		return true;
	}
	@Override
	public String toString() {
```

```
            return id.replace(" ", "_");
      }


}
```

Clase Trayecto

```
package us.lsi.graphs.cruceros;


import org.jgrapht.EdgeFactory;

import org.jgrapht.graph.DefaultWeightedEdge;


import us.lsi.graphs.StringEdgeFactory;


public class Trayecto extends DefaultWeightedEdge {


      public static Factoria factoria = new Factoria();


      private static final long serialVersionUID = 1L;



      static class Factoria implements StringEdgeFactory<Puerto,Trayecto>,
EdgeFactory<Puerto,Trayecto> {



      public Trayecto createEdge(Puerto c1, Puerto c2) {

        return new Trayecto(c1,c2);


      }



      public Trayecto createEdge(Puerto c1, Puerto c2, String[] formato) {

            return new Trayecto(c1,c2,formato);

      }

}
```

```
	private static  Double doubleValueOf(String[] formato, int pos) {
		if (pos<formato.length) {
			return Double.valueOf(formato[pos]);
		}
		return null;
	}

	private static  Integer intValueOf(String[] formato, int pos) {
		if (pos<formato.length) {
			return Integer.valueOf(formato[pos]);
		}
		return null;
	}

	private Puerto origen = null;
	private Puerto destino = null;
	private Double costeTrayecto =0.0;
	private Integer duracionEnDias = 0;
	private Integer nivelDeSeguridad = null;


	public Trayecto() {
		super();
	}

	public Trayecto(Trayecto d) {
	this(d.origen,d.destino,d.costeTrayecto,d.duracionEnDias,d.nivelDeSeguri
dad);
	}

	public Trayecto(Puerto origen, Puerto destino) {
```

```
		this.origen=origen;
		this.destino=destino;
	}

	public Trayecto(Puerto origen, Puerto destino, String[] formato) {
		this(origen,destino,
				doubleValueOf(formato,2),
				intValueOf(formato,3),
				intValueOf(formato,4));

		if (formato.length<5) { System.out.println(this + "->" + "Numero
de campos erroneos"); }
		if (this.getCosteTrayecto()>100) { System.out.println(this + "->"
+ "ERR[2]: coste"  + this.getCosteTrayecto()); }
		if (this.getDuracionEnDias()>5) { System.out.println(this + "->"
+ "ERR[3]: duracion"  + this.getDuracionEnDias()); }
		if (this.getNivelDeSeguridad()>3) { System.out.println(this + "-
>" + "ERR[4]: seguridad " + this.getNivelDeSeguridad()); }
	}

	public Trayecto(Puerto origen, Puerto destino, Double costeTrayecto,
			Integer duracionEnDias, Integer nivelDeSeguridad) {
		super();
		this.origen = origen;
		this.destino = destino;
		this.costeTrayecto = costeTrayecto;
		this.duracionEnDias = duracionEnDias;
		this.nivelDeSeguridad = nivelDeSeguridad;
	}

	public Puerto getOrigen() {
		return origen;
	}
```

```
	public Puerto getDestino() {
		return destino;
	}

	public Double getCosteTrayecto() {
		return costeTrayecto;
	}

	public Integer getDuracionEnDias() {
		return duracionEnDias;
	}

	public Integer getNivelDeSeguridad() {
		return nivelDeSeguridad;
	}

	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + ((destino == null) ? 0 :
destino.hashCode());
		result = prime * result + ((origen == null) ? 0 :
origen.hashCode());
		return result;
	}

	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
```

```
		if (getClass() != obj.getClass())
			return false;
		Trayecto other = (Trayecto) obj;
		if (destino == null) {
			if (other.destino != null)
				return false;
		} else if (!destino.equals(other.destino))
			return false;
		if (origen == null) {
			if (other.origen != null)
				return false;
		} else if (!origen.equals(other.origen))
			return false;
		return true;
	}

	@Override
	public String toString() {
		return "[origen=" + origen + ", destino=" + destino
				+ "]";
	}


}
```

Clase MapaDeNavegacion

```
package us.lsi.graphs.cruceros;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
```

```
import java.util.stream.Collectors;
import org.jgrapht.Graph;
import org.jgrapht.GraphPath;
import org.jgrapht.Graphs;
import org.jgrapht.UndirectedGraph;
import org.jgrapht.alg.ConnectivityInspector;
import org.jgrapht.alg.FloydWarshallShortestPaths;
import org.jgrapht.alg.KruskalMinimumSpanningTree;
import org.jgrapht.alg.VertexCovers;
import org.jgrapht.graph.SimpleGraph;
import org.jgrapht.graph.SimpleWeightedGraph;
import org.jgrapht.graph.WeightedMultigraph;

import us.lsi.graphs.GraphsReader;

public class MapaDeNavegacion {

	private Graph<Puerto, Trayecto> g = null;

	private MapaDeNavegacion() {
		this(null);
	}

	private MapaDeNavegacion(Graph<Puerto, Trayecto> g) {
		this.g = g;
	}

	public static MapaDeNavegacion read(String file) {
		// DONE 1

		UndirectedGraph<Puerto, Trayecto> g = new SimpleGraph<Puerto,
Trayecto>(Trayecto.factoria);
		g = (UndirectedGraph<Puerto, Trayecto>)
GraphsReader.newGraph(file, Puerto.factoria, Trayecto.factoria, g);
```

```
		return new MapaDeNavegacion(g);
	}

	public Graph<Puerto, Trayecto> getMapa() {
		// DONE 2
		return g;
	}

	public Puerto getPuertoById(String id) {
		// DONE 3
		return g.vertexSet().stream().filter(p ->
p.getId().equals(id)).findFirst().get();
	}

	public Set<Puerto> getRedDeControles() {
		// DONE 4
		return VertexCovers.find2ApproximationCover(getMapa());
	}

	public List<Set<Puerto>> getZonasDeInfluencia(int
limiteDeDiasXDesplazamiento) {
		// DONE 5

		List<Set<Puerto>> res = new ArrayList<Set<Puerto>>();

		UndirectedGraph<Puerto, Trayecto> grafo = new SimpleGraph<Puerto,
Trayecto>(Trayecto.factoria);

		getMapa().edgeSet().stream().filter(t -> (t.getDuracionEnDias()
<= limiteDeDiasXDesplazamiento))
				.forEach(t -> grafoAuxiliar(t, grafo));

		ConnectivityInspector<Puerto, Trayecto> ins = new
ConnectivityInspector<Puerto, Trayecto>(
				(UndirectedGraph<Puerto, Trayecto>) grafo);
```

```
            ins.connectedSets().stream().filter(s -> s.size() >= 3).forEach(s
-> res.add(s));


            return res;


      }


      private void grafoAuxiliar(Trayecto t, Graph<Puerto, Trayecto> graph) {
            graph.addVertex(t.getDestino());
            graph.addVertex(t.getOrigen());
            graph.addEdge(t.getDestino(), t.getOrigen(), t);
      }


      public Set<Trayecto> getRedDeFerris() {
            // DONE 6
            KruskalMinimumSpanningTree<Puerto, Trayecto> x = new
KruskalMinimumSpanningTree<>(getMapa());
            return x.getMinimumSpanningTreeEdgeSet();


      }


      public Integer getDuracionDeCrucero(List<Puerto> travesia) {
            // DONE 7


            Integer duracion = 0;
            for (Trayecto x : getMapa().edgeSet()) {
                  for (Puerto v : travesia) {
                        for (Puerto b : travesia) {
                              if (v.equals(x.getOrigen()) &&
b.equals(x.getDestino())) {
                                    duracion += x.getDuracionEnDias();
                              }
                        }
                  }
```

```
		}
		return duracion;

	}


	public Double getAtractivoMedio(List<Puerto> travesia) {
		// DONE 8
		Double suma = 0.0;
		Long numeroPuertos = travesia.stream().count();

		for (Puerto p : travesia) {
			suma += p.getAtractivo();
		}
		return suma / numeroPuertos;
	}

	public Boolean existeAlgunPuntoAbastecimiento(List<Puerto> travesia) {
		// DONE 9

		return travesia.stream().anyMatch(x ->
x.esPuntoDeAbastecimiento());

	}

	public Double getCosteXCrucero(List<Puerto> travesia) {
		// TODO 10

		Double tasaPuertos = 0.0;
		for (Puerto p : travesia) {
			tasaPuertos = tasaPuertos + p.getTasas();
		}

		Double costeTrayecto = 0.0;
```

```
		for (Trayecto x : getMapa().edgeSet()) {
			for (Puerto v : travesia) {
				for (Puerto b : travesia) {
					if (v.equals(x.getOrigen()) &&
b.equals(x.getDestino())) {
						costeTrayecto += x.getCosteTrayecto();
					}
				}
			}
		}
		return tasaPuertos + costeTrayecto;

	}

	public Set<Puerto> getPuertosConAeropuertosCercanos() {
		// DONE 11

		return g.vertexSet().stream()
				.filter(x -> x.getMinutosHastaElAeropuerto() < 61 &&
x.getMinutosHastaElAeropuerto() > 0)
				.collect(Collectors.toSet());
	}

	public Set<List<Puerto>> getCrucerosValidos() {
		// DONE 12

		Set<List<Puerto>> res = new HashSet<List<Puerto>>();
		Set<Puerto> aeropuertosCercanos = new
HashSet<Puerto>(this.getPuertosConAeropuertosCercanos());

		FloydWarshallShortestPaths<Puerto, Trayecto> caminos = new
FloydWarshallShortestPaths<Puerto, Trayecto>(
				getGrafoPesos(g));
```

```
            caminos.getShortestPaths().stream().forEach(caminosMinimos ->
res.add(caminosMinimos(caminosMinimos)));


            return res.stream().filter(

                        x -> aeropuertosCercanos.contains(x.get(0)) &&
aeropuertosCercanos.contains(x.get(x.size() - 1)))

                        .collect(Collectors.toSet());

      }


      private List<Puerto> caminosMinimos(GraphPath<Puerto, Trayecto>
caminosMinimos) {

            List<Puerto> res = new ArrayList<Puerto>();

            res = Graphs.getPathVertexList(caminosMinimos);

            return res;

      }


      private SimpleWeightedGraph<Puerto, Trayecto>
getGrafoPesos(Graph<Puerto, Trayecto> graph) {


            SimpleWeightedGraph<Puerto, Trayecto> res = new
SimpleWeightedGraph<Puerto, Trayecto>(Trayecto.factoria);

            Graphs.addAllVertices(res, graph.vertexSet());


            for (Trayecto t : graph.edgeSet()) {

                  res.setEdgeWeight(res.addEdge(graph.getEdgeSource(t),
graph.getEdgeTarget(t)), t.getDuracionEnDias());

            }

            return res;

      }


      public Set<List<Puerto>> getCrucerosFantasticos() {

            // DONE 13


            return getCrucerosValidos().stream()

                        .filter(x -> getDuracionDeCrucero(x) >= 9 &&
getDuracionDeCrucero(x) <= 12
```

```
                                        && existeAlgunPuntoAbastecimiento(x)
&& getAtractivoMedio(x) >= 2.3)
                            .collect(Collectors.toSet());
        }


}
```

4.-Test y Pruebas

Tengo problemas a la hora de copiar el código y no consigo que copie el formato original.

TestMapaDeNavegacion

```
package us.lsi.graphs.cruceros;

import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;

import org.jgrapht.Graph;
import org.jgrapht.ext.ComponentAttributeProvider;
import org.jgrapht.ext.EdgeNameProvider;
import org.jgrapht.ext.VertexNameProvider;

import us.lsi.graphs.GraphsFileExporter;

import com.google.common.collect.Sets;

public class TestMapaDeNavegacion {
       static public void main(String[] args) {
              MapaDeNavegacion                    mn                    =
MapaDeNavegacion.read("ficheros//cruceros.txt");
              Graph<Puerto,Trayecto> g = mn.getMapa();

              Set<Puerto> ms = Sets.newHashSet();
              for (Puerto p:g.vertexSet()) {

              }
         for (Trayecto e:g.edgeSet()) {
```

```
            }

        GraphsFileExporter.<Puerto,Trayecto>saveFile(g,
"ficheros//salida6.gsv", ms,new HashSet<Trayecto>());
        /*
              GraphsFileExporter.<Puerto,Trayecto>saveFile(g,
"ficheros//salida6.gsv",
                           new IdPuertos(),
                           new NamePuertos(),
                           new NameTrayectos(),
                           new EtiquetasPuertos(),
                           new EtiquetasTrayectos());
           */

              Set<Puerto> s = mn.getRedDeControles();
              System.out.println("RED DE CONTROLES: " + s +  ". SIZE: "
+ s.size());
              System.out.println();

              int cont = 0;
              System.out.println("ZONAS DE INFLUENCIA(1): ");
              for (Set<Puerto> ss: mn.getZonasDeInfluencia(1)) {
                     System.out.println((cont++) + " -> " + ss);
              }
              System.out.println();

              System.out.println("RED        DE        FERRIS:"        +
mn.getRedDeFerris());
              System.out.println();

              System.out.println("PUERTOS CON AEROPUERTOS CERCANOS: " +
mn.getPuertosConAeropuertosCercanos());
              System.out.println();

              System.out.println("CRUCEROS DE FANTASTICO");
              System.out.println("======================");
              for (List<Puerto> c:mn.getCrucerosFantasticos()) {
                     System.out.println(c + " -> " +
                        "(a=" + mn.getAtractivoMedio(c) + ", " +
                              "d=" + mn.getDuracionDeCrucero(c) + ", " +
                         "l="  +  mn.existeAlgunPuntoAbastecimiento(c)  +
", " +
                         "c=" + mn.getCosteXCrucero(c) +
                       ")") ;
              }
              System.out.println();

       }

      private       static       class       IdPuertos       implements
VertexNameProvider<Puerto> {

              @Override
              public String getVertexName(Puerto arg0) {
                     // TODO Auto-generated method stub
                     return "\"" + arg0.getId() + "\"";
```

```
		}

	}

	private	static	class	NamePuertos	implements
VertexNameProvider<Puerto> {

		@Override
		public String getVertexName(Puerto v) {
			// TODO Auto-generated method stub
			//return arg0.getId();

			Map<String, String> map = new HashMap<>();
			map.put("t", String.valueOf(v.getTasas()));
			map.put("a", String.valueOf(v.getAtractivo()));
			map.put("c", String.valueOf(v.getCalado()));
			map.put("ma",
String.valueOf(v.getMinutosHastaElAeropuerto()));
			map.put("l",
String.valueOf(v.esPuntoDeAbastecimiento()));
			return v.getId() + "/" + v.getPais() + "\\n" +
map.toString();
		}

	}

	private	static	class	NameTrayectos	implements
EdgeNameProvider<Trayecto> {

		@Override
		public String getEdgeName(Trayecto e) {
			// TODO Auto-generated method stub
			//return arg0.getOrigen().getId() + " -- " +
arg0.getDestino().getId();
			Map<String, String> map = new HashMap<>();
			map.put("c", String.valueOf(e.getCosteTrayecto()));
			map.put("d", String.valueOf(e.getDuracionEnDias()));
			map.put("s",
String.valueOf(e.getNivelDeSeguridad()));
			return map.toString();
		}

	}

    private	static	class	EtiquetasPuertos	implements
ComponentAttributeProvider<Puerto>{

		public EtiquetasPuertos() {
		}

		@Override
		public Map<String, String> getComponentAttributes(Puerto
v) {
			Map<String, String> map = new HashMap<>();
				map.put("tasas",
String.valueOf(v.getTasas()));
```

```
                    map.put("atractivo",
String.valueOf(v.getAtractivo()));
                    map.put("calado", String.valueOf(v.getCalado()));
                    map.put("maeropueto",
String.valueOf(v.getMinutosHastaElAeropuerto()));
                    map.put("abastecimiento",
String.valueOf(v.esPuntoDeAbastecimiento()));
                    //map.put("style", "dotted");
                    return map;
             }

      }

      private    static    class    EtiquetasTrayectos    implements
ComponentAttributeProvider<Trayecto>{

             public EtiquetasTrayectos() {
             }

             @Override
             public Map<String, String> getComponentAttributes(Trayecto
e) {
                    Map<String, String> map = new HashMap<>();
                    map.put("c", String.valueOf(e.getCosteTrayecto()));
                    map.put("d", String.valueOf(e.getDuracionEnDias()));
                    map.put("s",
String.valueOf(e.getNivelDeSeguridad()));
                    //map.put("style", "dotted");
                    return map;
             }

      }
}
```

Salida correspondiente

```
RED DE CONTROLES: [Oslo, Barcelona, Bergen, Livorno, Mesina,
Copenhague, Rostock, Katakolon, Villefranche, Rodas, Flam, Dubrovnik,
Mykonos, Southampton, Cannes, Rijeka, La_Valletta, Estocolmo, Kotor,
Mytilene, Civitavecchia, Cadiz, Stavanger, Cabo_Norte, El_Pireo,
Haifa, Lisboa, Tallin, La_Spezia, Marsella, Palermo, Alesund,
Amsterdam, Iraklion, Venecia, Molde, Helsinki, Santorini]. SIZE: 38

ZONAS DE INFLUENCIA(1):
0 -> [Haugesund, Oslo, Bergen, Copenhague, Stavanger, Cabo_Norte,
Flam, Tallin, Trondheim, Alesund, Kristiansand, Amsterdam, Molde]
1 -> [Barcelona, La_Valletta, Livorno, Civitavecchia, Mesina, Cadiz,
Napoles, Villefranche, La_Spezia, Marsella, Palermo, Cannes,
Palma_de_Mallorca]
2 -> [Rijeka, Dubrovnik, Kotor, Venecia, Zadar, Split]
3 -> [Mykonos, Iraklion, Mytilene, Limassol, El_Pireo, Katakolon,
Haifa, Santorini, Rodas]
4 -> [Estocolmo, San_Petersburgo, Helsinki]

RED DE FERRIS:[[origen=Katakolon, destino=Mytilene], [origen=Mykonos,
destino=Rodas], [origen=Kristiansand, destino=Oslo],
```

[origen=Copenhague, destino=Oslo], [origen=La_Spezia, destino=Civitavecchia], [origen=Cadiz, destino=Barcelona], [origen=Cannes, destino=Livorno], [origen=Oslo, destino=Amsterdam], [origen=Barcelona, destino=Napoles], [origen=Alesund, destino=Molde], [origen=Cadiz, destino=Palma_de_Mallorca], [origen=Civitavecchia, destino=Mesina], [origen=La_Valletta, destino=Mykonos], [origen=Flam, destino=Trondheim], [origen=Cadiz, destino=Lisboa], [origen=Haugesund, destino=Flam], [origen=Barcelona, destino=Marsella], [origen=Rijeka, destino=Venecia], [origen=Venecia, destino=Zadar], [origen=Marsella, destino=Villefranche], [origen=Napoles, destino=Santorini], [origen=Iraklion, destino=Palermo], [origen=Oslo, destino=Warnemunde], [origen=Dubrovnik, destino=Rijeka], [origen=Haifa, destino=Katakolon], [origen=Cadiz, destino=Mesina], [origen=Flam, destino=Stavanger], [origen=Estocolmo, destino=Copenhague], [origen=Cannes, destino=Palma_de_Mallorca], [origen=Kotor, destino=Santorini], [origen=Helsinki, destino=Estocolmo], [origen=Dubrovnik, destino=El_Pireo], [origen=El_Pireo, destino=Barcelona], [origen=El_Pireo, destino=Mykonos], [origen=Tallin, destino=Cabo_Norte], [origen=Flam, destino=Alesund], [origen=Rostock, destino=Stavanger], [origen=Amsterdam, destino=Flam], [origen=Bergen, destino=Southampton], [origen=El_Pireo, destino=Limassol], [origen=Molde, destino=Bergen], [origen=El_Pireo, destino=Iraklion], [origen=Lisboa, destino=Southampton], [origen=Split, destino=Kotor], [origen=Limassol, destino=Haifa], [origen=San_Petersburgo, destino=Helsinki], [origen=Cabo_Norte, destino=Bergen]]

PUERTOS CON AEROPUERTOS CERCANOS: [Lisboa, Barcelona, Southampton, Estocolmo, Amsterdam, San_Petersburgo, Palma_de_Mallorca, Copenhague, Venecia, Cadiz, Napoles]

CRUCEROS DE FANTASTICO
==========
[Venecia, Zadar, El_Pireo, Barcelona, Cadiz, Lisboa] -> (a=3.0, d=10, l=true, c=340.0)
[Estocolmo, Copenhague, Oslo, Amsterdam, Palermo, Civitavecchia, Livorno, Marsella, Barcelona] -> (a=2.6666666666666665, d=11, l=true, c=590.0)
[Palma_de_Mallorca, Cannes, Livorno, Civitavecchia, Palermo, Amsterdam, Oslo, Copenhague] -> (a=2.625, d=9, l=true, c=515.0)
[Barcelona, Marsella, Livorno, Civitavecchia, Palermo, Amsterdam, Oslo, Copenhague, Estocolmo] -> (a=2.6666666666666665, d=11, l=true, c=590.0)
[Estocolmo, Copenhague, Oslo, Amsterdam, Palermo, Civitavecchia, Napoles] -> (a=2.857142857142857, d=9, l=true, c=425.0)
[Estocolmo, Helsinki, San_Petersburgo, Southampton, Lisboa, Cadiz, Palma_de_Mallorca] -> (a=2.5714285714285716, d=10, l=true, c=380.0)
[Venecia, Zadar, El_Pireo, Iraklion, Palermo, Amsterdam] -> (a=2.8333333333333335, d=9, l=true, c=380.0)
[San_Petersburgo, Southampton, Lisboa, Cadiz, Barcelona] -> (a=2.4, d=9, l=true, c=305.0)
[Cadiz, Lisboa, Southampton, San_Petersburgo, Helsinki, Estocolmo] -> (a=2.5, d=9, l=true, c=340.0)
[Estocolmo, Helsinki, San_Petersburgo, Southampton, Lisboa, Cadiz] -> (a=2.5, d=9, l=true, c=340.0)
[Barcelona, Marsella, Livorno, Civitavecchia, Palermo, Amsterdam, Oslo, Copenhague] -> (a=2.625, d=9, l=true, c=530.0)

```
[Copenhague, Oslo, Amsterdam, Palermo, Civitavecchia, Mesina, Cadiz] -
> (a=2.857142857142857, d=9, l=true, c=445.0)
[Copenhague, Oslo, Amsterdam, Palermo, Civitavecchia, Livorno,
Marsella, Barcelona] -> (a=2.625, d=9, l=true, c=530.0)
[Napoles, Civitavecchia, Palermo, Amsterdam, Oslo, Copenhague,
Estocolmo] -> (a=2.857142857142857, d=9, l=true, c=425.0)
[Copenhague, Oslo, Amsterdam, Palermo, Civitavecchia, Livorno, Cannes,
Palma_de_Mallorca] -> (a=2.625, d=9, l=true, c=515.0)
[Barcelona, Cadiz, Lisboa, Southampton, San_Petersburgo] -> (a=2.4,
d=9, l=true, c=305.0)
[Copenhague, Oslo, Amsterdam, Palermo, Iraklion, El_Pireo, Zadar,
Venecia] -> (a=2.875, d=11, l=true, c=530.0)
[Southampton, Lisboa, Cadiz, Barcelona, El_Pireo, Zadar, Venecia] ->
(a=2.7142857142857144, d=12, l=true, c=400.0)
[Napoles, Civitavecchia, Palermo, Amsterdam, Oslo, Copenhague,
Estocolmo, Helsinki, San_Petersburgo] -> (a=2.7777777777777777, d=11,
l=true, c=545.0)
[Amsterdam, Palermo, Iraklion, El_Pireo, Zadar, Venecia] ->
(a=2.8333333333333335, d=9, l=true, c=380.0)
[Lisboa, Cadiz, Barcelona, El_Pireo, Zadar, Venecia] -> (a=3.0, d=10,
l=true, c=340.0)
[Palma_de_Mallorca, Cadiz, Lisboa, Southampton, San_Petersburgo,
Helsinki, Estocolmo] -> (a=2.5714285714285716, d=10, l=true, c=380.0)
[San_Petersburgo, Helsinki, Estocolmo, Copenhague, Oslo, Amsterdam,
Palermo, Civitavecchia, Napoles] -> (a=2.7777777777777777, d=11,
l=true, c=545.0)
[Venecia, Zadar, El_Pireo, Barcelona, Cadiz, Lisboa, Southampton] ->
(a=2.7142857142857144, d=12, l=true, c=400.0)
[Venecia, Zadar, El_Pireo, Iraklion, Palermo, Amsterdam, Oslo,
Copenhague] -> (a=2.875, d=11, l=true, c=530.0)
[Cadiz, Mesina, Civitavecchia, Palermo, Amsterdam, Oslo, Copenhague] -
> (a=2.857142857142857, d=9, l=true, c=445.0)
```

Test de los demás métodos

```
package us.lsi.graphs.cruceros;

import java.util.ArrayList;
import java.util.List;
import org.jgrapht.Graph;

public class Test {

      public static void main(String[] args) {

            // TEST METODO 2

            MapaDeNavegacion                    mn                    =
MapaDeNavegacion.read("ficheros//cruceros.txt");
            Graph<Puerto, Trayecto> g = mn.getMapa();
            System.out.println("Test Metodo 2)" + g);

            // TEST METODO 3
```

```
            System.out.println("Test        Metodo        3)"        +
mn.getPuertoById("Alesund"));


            // TEST METODO 7

            List<Puerto> travesia = new ArrayList<Puerto>();
            travesia.add(mn.getPuertoById("Barcelona"));
            travesia.add(mn.getPuertoById("Napoles"));
            travesia.add(mn.getPuertoById("Santorini"));
            travesia.add(mn.getPuertoById("Rodas"));
            System.out.println("Test        Metodo        7)"        +
mn.getDuracionDeCrucero(travesia));

            // TEST METODO 8

            List<Puerto> travesia8 = new ArrayList<Puerto>();
            travesia8.add(mn.getPuertoById("Barcelona"));
            travesia8.add(mn.getPuertoById("Napoles"));
            travesia8.add(mn.getPuertoById("Santorini"));
            travesia8.add(mn.getPuertoById("Rodas"));
            System.out.println("Test        Metodo        8)"        +
mn.getAtractivoMedio(travesia8));

            // TEST METODO 9

            List<Puerto> travesia2 = new ArrayList<Puerto>();
            travesia2.add(mn.getPuertoById("Estocolmo"));
            travesia2.add(mn.getPuertoById("Napoles"));
            travesia2.add(mn.getPuertoById("Santorini"));
            travesia2.add(mn.getPuertoById("Rodas"));
            System.out.println("Test        Metodo        9)"        +
mn.existeAlgunPuntoAbastecimiento(travesia2));

            // TEST METODO 10

            List<Puerto> travesia1 = new ArrayList<Puerto>();
            travesia1.add(mn.getPuertoById("Barcelona"));
            travesia1.add(mn.getPuertoById("Napoles"));
            travesia1.add(mn.getPuertoById("Santorini"));
            travesia1.add(mn.getPuertoById("El Pireo"));
            System.out.println("Test        Metodo        10)"        +
mn.getCosteXCrucero(travesia1));


            // TEST METODO 12

            System.out.println("Test        Metodo        12)"        +
mn.getCrucerosValidos().size());

      }
```

}

Salida correspondiente

```
Test Metodo 2)([Alesund, Amsterdam, Barcelona, Bergen, Cabo_Norte,
Cadiz, Cannes, Civitavecchia, Copenhague, Dubrovnik, El_Pireo,
Estocolmo, Flam, Haifa, Haugesund, Helsinki, Iraklion, Katakolon,
Kotor, Kristiansand, La_Spezia, La_Valletta, Limassol, Lisboa,
Livorno, Marsella, Mesina, Molde, Mykonos, Mytilene, Napoles, Oslo,
Palermo, Palma_de_Mallorca, Rijeka, Rodas, Rostock, San_Petersburgo,
Santorini, Southampton, Split, Stavanger, Tallin, Trondheim, Venecia,
Villefranche, Warnemunde, Zadar], [[origen=Alesund,
destino=Molde]={Alesund,Molde}, [origen=Amsterdam,
destino=Flam]={Amsterdam,Flam}, [origen=Barcelona,
destino=Marsella]={Barcelona,Marsella}, [origen=Barcelona,
destino=Napoles]={Barcelona,Napoles}, [origen=Bergen,
destino=Southampton]={Bergen,Southampton}, [origen=Cabo_Norte,
destino=Bergen]={Cabo_Norte,Bergen}, [origen=Cadiz,
destino=Barcelona]={Cadiz,Barcelona}, [origen=Cadiz,
destino=Lisboa]={Cadiz,Lisboa}, [origen=Cadiz,
destino=Mesina]={Cadiz,Mesina}, [origen=Cadiz,
destino=Palma_de_Mallorca]={Cadiz,Palma_de_Mallorca}, [origen=Cannes,
destino=Livorno]={Cannes,Livorno}, [origen=Cannes,
destino=Palma_de_Mallorca]={Cannes,Palma_de_Mallorca},
[origen=Civitavecchia, destino=Mesina]={Civitavecchia,Mesina},
[origen=Civitavecchia, destino=Livorno]={Civitavecchia,Livorno},
[origen=Civitavecchia, destino=Napoles]={Civitavecchia,Napoles},
[origen=Copenhague, destino=Oslo]={Copenhague,Oslo},
[origen=Dubrovnik, destino=El_Pireo]={Dubrovnik,El_Pireo},
[origen=Dubrovnik, destino=Rijeka]={Dubrovnik,Rijeka},
[origen=El_Pireo, destino=Barcelona]={El_Pireo,Barcelona},
[origen=El_Pireo, destino=Iraklion]={El_Pireo,Iraklion},
[origen=El_Pireo, destino=Limassol]={El_Pireo,Limassol},
[origen=El_Pireo, destino=Mykonos]={El_Pireo,Mykonos},
[origen=Estocolmo, destino=Copenhague]={Estocolmo,Copenhague},
[origen=Flam, destino=Alesund]={Flam,Alesund}, [origen=Flam,
destino=Stavanger]={Flam,Stavanger}, [origen=Flam,
destino=Trondheim]={Flam,Trondheim}, [origen=Haifa,
destino=Katakolon]={Haifa,Katakolon}, [origen=Haugesund,
destino=Flam]={Haugesund,Flam}, [origen=Helsinki,
destino=Estocolmo]={Helsinki,Estocolmo}, [origen=Iraklion,
destino=Palermo]={Iraklion,Palermo}, [origen=Katakolon,
destino=Mytilene]={Katakolon,Mytilene}, [origen=Kotor,
destino=Santorini]={Kotor,Santorini}, [origen=Kristiansand,
destino=Oslo]={Kristiansand,Oslo}, [origen=La_Spezia,
destino=Civitavecchia]={La_Spezia,Civitavecchia}, [origen=La_Valletta,
destino=Mykonos]={La_Valletta,Mykonos}, [origen=Limassol,
destino=Haifa]={Limassol,Haifa}, [origen=Lisboa,
destino=Southampton]={Lisboa,Southampton}, [origen=Livorno,
destino=Marsella]={Livorno,Marsella}, [origen=Marsella,
destino=Cannes]={Marsella,Cannes}, [origen=Marsella,
destino=Villefranche]={Marsella,Villefranche}, [origen=Mesina,
destino=La_Valletta]={Mesina,La_Valletta}, [origen=Molde,
destino=Bergen]={Molde,Bergen}, [origen=Mykonos,
```

```
destino=Dubrovnik]={Mykonos,Dubrovnik}, [origen=Mykonos,
destino=Rodas]={Mykonos,Rodas}, [origen=Mytilene,
destino=El_Pireo]={Mytilene,El_Pireo}, [origen=Napoles,
destino=Santorini]={Napoles,Santorini}, [origen=Oslo,
destino=Amsterdam]={Oslo,Amsterdam}, [origen=Oslo,
destino=Warnemunde]={Oslo,Warnemunde}, [origen=Palermo,
destino=Civitavecchia]={Palermo,Civitavecchia},
[origen=Palma_de_Mallorca,
destino=Barcelona]={Palma_de_Mallorca,Barcelona}, [origen=El_Pireo,
destino=Napoles]={El_Pireo,Napoles}, [origen=Rijeka,
destino=Venecia]={Rijeka,Venecia}, [origen=Rodas,
destino=Mytilene]={Rodas,Mytilene}, [origen=Rodas,
destino=Santorini]={Rodas,Santorini}, [origen=Rostock,
destino=Stavanger]={Rostock,Stavanger}, [origen=San_Petersburgo,
destino=Helsinki]={San_Petersburgo,Helsinki}, [origen=Santorini,
destino=El_Pireo]={Santorini,El_Pireo}, [origen=Southampton,
destino=Stavanger]={Southampton,Stavanger}, [origen=Split,
destino=Kotor]={Split,Kotor}, [origen=Stavanger,
destino=Haugesund]={Stavanger,Haugesund}, [origen=Stavanger,
destino=Kristiansand]={Stavanger,Kristiansand}, [origen=Stavanger,
destino=Molde]={Stavanger,Molde}, [origen=Tallin,
destino=Cabo_Norte]={Tallin,Cabo_Norte}, [origen=Trondheim,
destino=Cabo_Norte]={Trondheim,Cabo_Norte}, [origen=Venecia,
destino=Dubrovnik]={Venecia,Dubrovnik}, [origen=Venecia,
destino=Split]={Venecia,Split}, [origen=Venecia,
destino=Zadar]={Venecia,Zadar}, [origen=Villefranche,
destino=La_Spezia]={Villefranche,La_Spezia}, [origen=Warnemunde,
destino=Tallin]={Warnemunde,Tallin}, [origen=Zadar,
destino=El_Pireo]={Zadar,El_Pireo}, [origen=Southampton,
destino=San_Petersburgo]={Southampton,San_Petersburgo},
[origen=Amsterdam, destino=Palermo]={Amsterdam,Palermo}])
Test Metodo 3)Alesund
Test Metodo 7)5
Test Metodo 8)2.75
Test Metodo 9)true
Test Metodo 10)280.0
Test Metodo 12)110
```