Tabla de contenido

Ejercicio1

Ejercicio1.java

```
package ejercicios;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.function.Predicate;
import java.util.stream.Collectors;

import org.jgrapht.Graph;
import org.jgrapht.Graphs;
import org.jgrapht.alg.interfaces.VertexCoverAlgorithm.VertexCover;
import org.jgrapht.alg.vertexcover.GreedyVCImpl;

import datos.Personas;
import datos.Rela;
import datos.Relacion;

import us.lsi.colors.GraphColors;
import us.lsi.colors.GraphColors.Color;

import us.lsi.graphs.views.SubGraphView;

public class Ejercicio1 {
	//	Obtenga una vista del grafo que sólo incluya las personas cuyos padres aparecen
	//	en el grafo, y ambos han nacido en la misma ciudad y en el mismo año. Muestre
	//	el grafo configurando su apariencia de forma que se resalten los vértices y las
	//	aristas de la vista.
	public static void apartadoA(String file, Graph<Personas, Relacion> g,
Predicate<Personas> pv, Predicate<Relacion> pa, String nombreVista) {
		Graph<Personas, Relacion> vista = SubGraphView.of(g, pv, pa);

		String fileRes = "resultados/ejercicio1/" + file + nombreVista + ".gv";
		GraphColors.toDot(g, fileRes,
				v->v.nombre(),e-> "",
				v-> GraphColors.colorIf(Color.blue, vista.vertexSet().contains(v)),
				e->  GraphColors.colorIf(Color.black, vista.edgeSet().contains(e))
				);
		System.out.println("Se ha generado " + fileRes);

	}

	//	Implemente un algoritmo que dada una persona devuelva un conjunto con todos
	//	sus ancestros que aparecen en el grafo. Muestre el grafo configurando su
	//	apariencia de forma que se resalte la persona de un color y sus ancestros de otro.

	public static List<Personas> apartadoB_Aux(Graph<Personas, Relacion> g,Personas persona,
List<Personas> acum){
		if(Graphs.vertexHasPredecessors(g, persona)) {
			for(Personas p : Graphs.predecessorListOf(g, persona)) {
				acum.add(p);
				apartadoB_Aux(g, p, acum);
			}
		}
		return acum;
	}

	private static Color asignaColor(Personas v, List<Personas> ls,Personas persona) {
```

```
			return ls.contains(v)? Color.blue : v == persona? Color.red : Color.black;
		}

		public static void apartadoB(String file, Graph<Personas, Relacion> g, String
nombrePersona, String nombreVista) {
			Personas persona = g.vertexSet().stream().filter(v ->
v.nombre().equals(nombrePersona)).findFirst().orElse(null);
			List<Personas> res = apartadoB_Aux(g, persona, new ArrayList<>());

			System.out.println(res);

			String fileRes = "resultados/ejercicio1/" + file + nombreVista + ".gv";
			GraphColors.toDot(g, fileRes,
					v->v.nombre(),e-> e.hijo().toString(),
					v-> GraphColors.color(asignaColor(v, res, persona)),
					e-> GraphColors.color(Color.black)
					);

			System.out.println("Se ha generado " + fileRes);


		}

		//		Implemente un algoritmo que dadas dos personas devuelva un valor entre los
		//		posibles del enumerado {Hermanos, Primos, Otros} en función de si son
		//		hermanos, primos hermanos, o ninguna de las dos cosas. Tenga en cuenta que
2
		//		personas son hermanas en caso de que tengan al padre o a la madre en común,
y
		//		primas en caso de tener al menos un abuelo/a en común.

		public static Rela apartadoC(String nombrePersona1, String nombrePersona2,
Graph<Personas, Relacion> g) {
			Personas persona1 = g.vertexSet().stream().filter(v ->
v.nombre().equals(nombrePersona1)).findFirst().orElse(null);
			Personas persona2 = g.vertexSet().stream().filter(v ->
v.nombre().equals(nombrePersona2)).findFirst().orElse(null);

			if(!Collections.disjoint(Graphs.predecessorListOf(g, persona1),
Graphs.predecessorListOf(g, persona2))) return Rela.Hermanos;
			else {
				List<Personas> abuelos1 = new ArrayList<>();
				Graphs.predecessorListOf(g, persona1).forEach(x ->
abuelos1.addAll(Graphs.predecessorListOf(g, x)));
				List<Personas> abuelos2 = new ArrayList<>();
				Graphs.predecessorListOf(g, persona2).forEach(x ->
abuelos2.addAll(Graphs.predecessorListOf(g, x)));
				if(!Collections.disjoint(abuelos1, abuelos2)) return Rela.Primos;
				else return Rela.Otros;
			}
		}

		//		Implemente un algoritmo que devuelva un conjunto con todas las personas que
		//		tienen hijos/as con distintas personas. Muestre el grafo configurando su
apariencia
		//		de forma que se resalten las personas de dicho conjunto.

		public static void apartadoD(String file, Graph<Personas, Relacion> g, String
nombreVista){
			Predicate<Personas> pv1 = v -> {
				List<Personas> lista = new ArrayList<>();
```

```
                for(Personas y : Graphs.successorListOf(g, v)) {
                        for(Personas x : Graphs.predecessorListOf(g, y)) {
                                lista.add(x);
                        }
                }
                return lista.stream().distinct().count()>2;
        };

        System.out.println(g.vertexSet().stream().filter(x ->
pv1.test(x)).collect(Collectors.toSet()));


        String fileRes = "resultados/ejercicio1/" + file + nombreVista + ".gv";
        GraphColors.toDot(g, fileRes,
                        v->v.nombre(),e-> e.hijo().toString(),
                        v-> GraphColors.colorIf(Color.blue, pv1.test(v)),
                        e-> GraphColors.color(Color.black)
                        );
        System.out.println("Se ha generado " + fileRes);

    }

    //          Se desea seleccionar el conjunto mínimo de personas para que se cubran
todas
    //          las relaciones existentes. Implemente un método que devuelva dicho
conjunto.
    //          Muestre el grafo configurando su apariencia de forma que se resalten las
personas
    //          de dicho conjunto.

    public static void apartadoE(Graph<Personas, Relacion> g, String file, String
nombreVista){
        Graph<Personas, Relacion> g2 = Graphs.undirectedGraph(g);

        GreedyVCImpl<Personas, Relacion> algA = new GreedyVCImpl<>(g2);
        VertexCover<Personas> conjunto = algA.getVertexCover();
        System.out.println("El conjunto esta compuesto por " + conjunto.size() + "
personas: ");
        conjunto.forEach(c -> System.out.println(c));

        String fileRes = "resultados/ejercicio1/" + file + nombreVista + ".gv";
        GraphColors.toDot(g, fileRes,
                        v->v.nombre(),e-> e.hijo().toString(),
                        v-> GraphColors.colorIf(Color.blue, conjunto.contains(v)),
                        e-> GraphColors.color(Color.black)
                        );
        System.out.println("Se ha generado " + fileRes);

    }


}
```

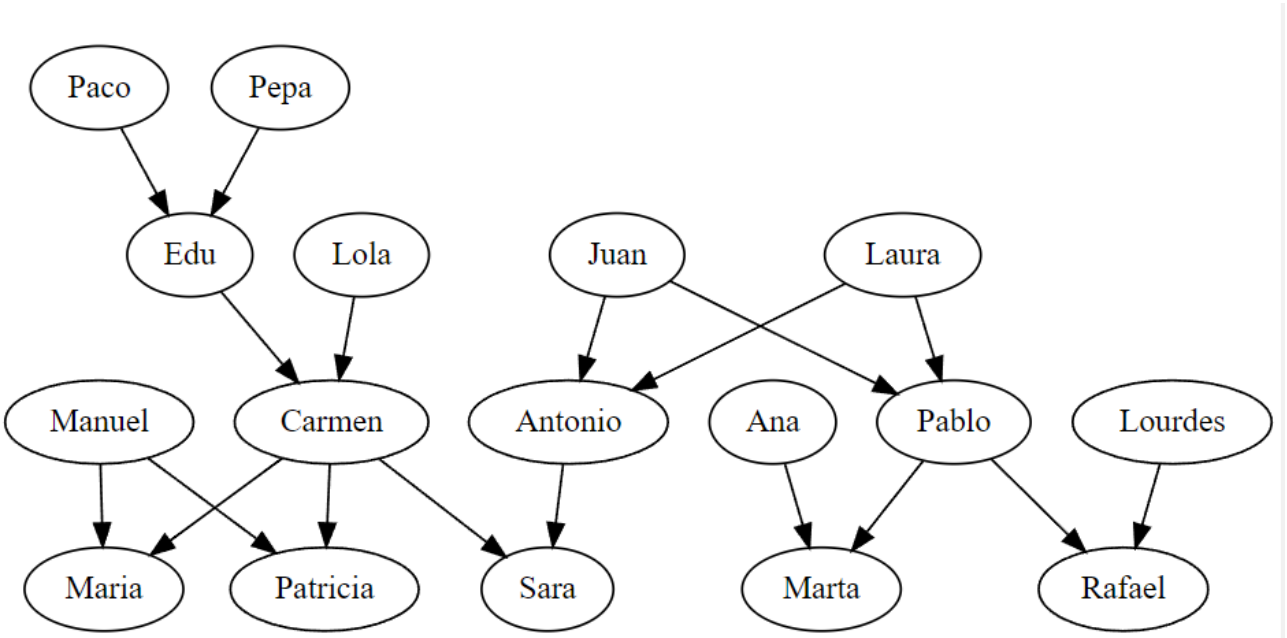
TestEjercicio1.java

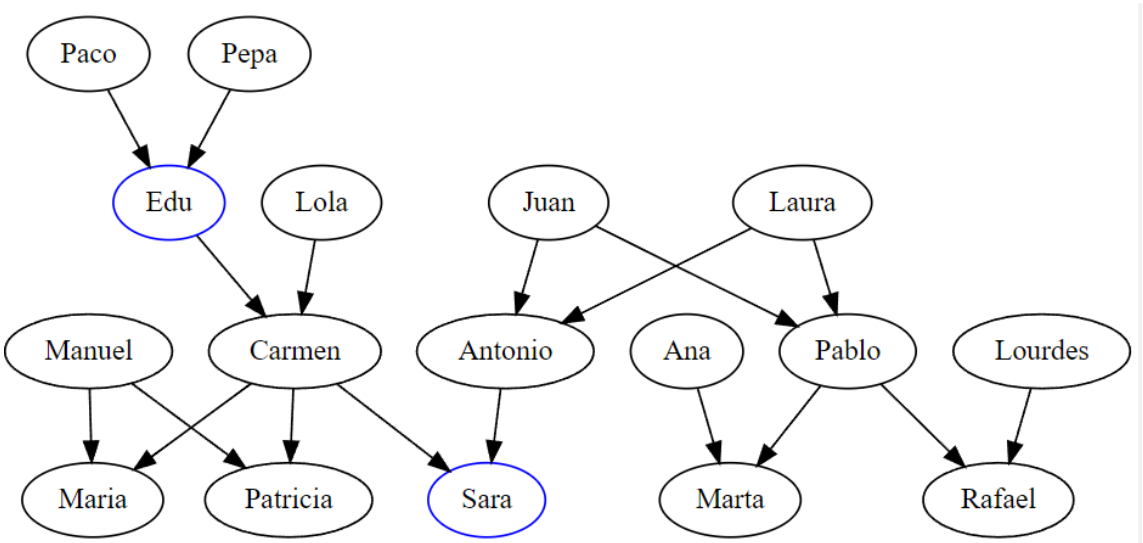
```
package tests;

import java.util.function.Predicate;

import org.jgrapht.Graph;
import org.jgrapht.Graphs;

import datos.Personas;
import datos.Relacion;
import ejercicios.Ejercicio1;
import us.lsi.colors.GraphColors;
import us.lsi.colors.GraphColors.Color;
import us.lsi.graphs.Graphs2;
import us.lsi.graphs.GraphsReader;

public class TestEjercicio1 {

	public static void main(String[] args) {
		testEjercicio1("PI3E1A_DatosEntrada");
		testEjercicio1("PI3E1B_DatosEntrada");

	}

	public static void testEjercicio1(String file) {

		//leer los datos de entrada
		Graph<Personas, Relacion> g =
				GraphsReader.newGraph("ficheros/"+file+".txt",
						Personas::ofFormat, Relacion::ofFormat,
Graphs2::simpleDirectedGraph);

		//Generar el archivo gv
		String fileRes = "resultados/ejercicio1/" + file + ".gv";
		GraphColors.toDot(g, fileRes,
				v -> v.nombre(), e-> e.hijo().toString(),
				v-> GraphColors.color(Color.black),
				e-> GraphColors.color(Color.black)
				);

		//variables dependiendo del fichero que se use
		String nombrePersona0 = "Maria";
		String nombrePersona2 = "Carmen";
		String nombrePersona1 = "Rafael";
		String nombrePersona3 = "Maria";
		String nombrePersona4 = "Patricia";
		String nombrePersona5 = "Sara";

		if(file == "PI3E1B_DatosEntrada") {
			nombrePersona0 = "Raquel";
			nombrePersona2 = "Laura";
			nombrePersona1 = "Raquel";
			nombrePersona3 = "Angela";
			nombrePersona4 = "Julia";
			nombrePersona5 = "Alvaro";
		}


		//Apartado a)
```

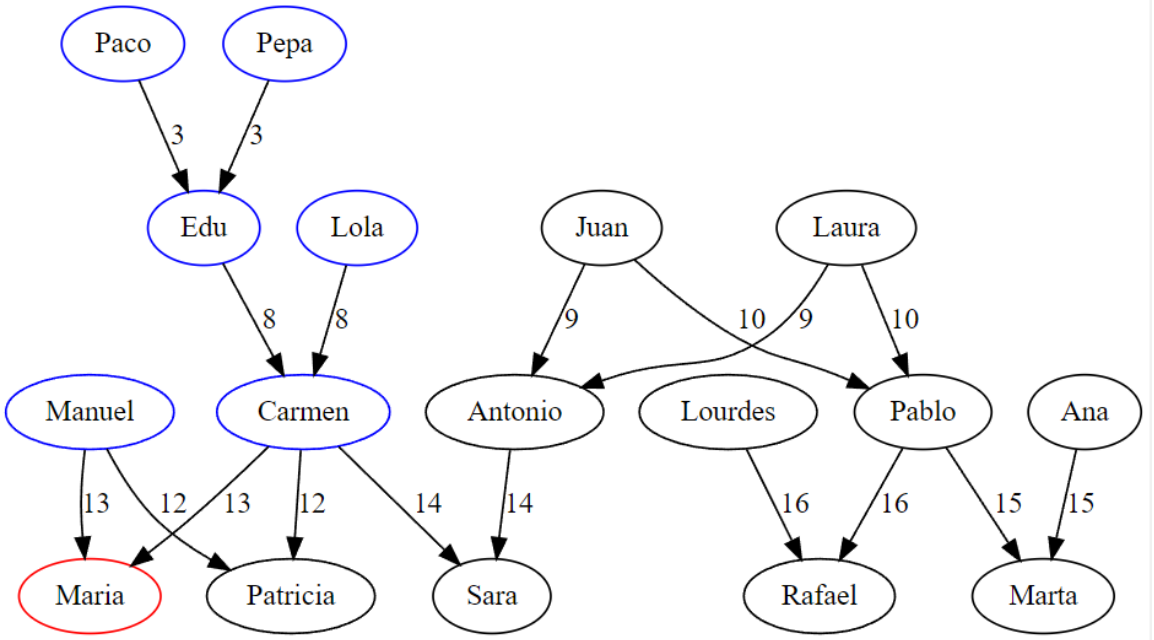
```
            Predicate<Personas> pv1 = c-> g.edgesOf(c).stream().filter(x -> x.hijo() ==
c.id()).count()==2 //comprueba que están sus dos padres
                        && Graphs.predecessorListOf(g,
c).stream().map(Personas::lugarNacimiento).distinct().limit(2).count() <= 1 //compruebo que han
nacido en el mismo sitio
                        && Graphs.predecessorListOf(g,
c).stream().map(Personas::anyo).distinct().limit(2).count() <= 1; //compruebo que han nacido el
mismo año
            Predicate<Relacion> pa1 = Relacion -> true;

            System.out.println("APARTADO A:");
            Ejercicio1.apartadoA(file, g, pv1, pa1, "Apartado A");
            System.out.println(" ");
            //Apartado b)
            System.out.println("APARTADO B:");
            Ejercicio1.apartadoB(file, g, nombrePersona0, "Apartado B");
            System.out.println(" ");

            //Apartado c)
            System.out.println("APARTADO C:");

            System.out.println(nombrePersona1 + " y " + nombrePersona2 + " son " +
Ejercicio1.apartadoC(nombrePersona1, nombrePersona2, g));
            System.out.println(nombrePersona1 + " y " + nombrePersona5 + " son " +
Ejercicio1.apartadoC(nombrePersona1, nombrePersona5, g));
            System.out.println(nombrePersona3 + " y " + nombrePersona4 + " son " +
Ejercicio1.apartadoC(nombrePersona3, nombrePersona4, g));

            System.out.println(" ");

            //Apartado d)
            System.out.println("APARTADO D:");
            Ejercicio1.apartadoD(file, g, "Apartado D");
            System.out.println(" ");

            //Apartado e)
            System.out.println("APARTADO E:");
            Ejercicio1.apartadoE(g,file, "Apartado E");
            System.out.println(" ");


      }

      //obtener persona por id
      static public Personas getPersonaById(Integer id, Graph<Personas, Relacion> g ) {
            return g.vertexSet().stream().filter(v ->
v.id().equals(id)).findFirst().orElse(null);
      }
}
```

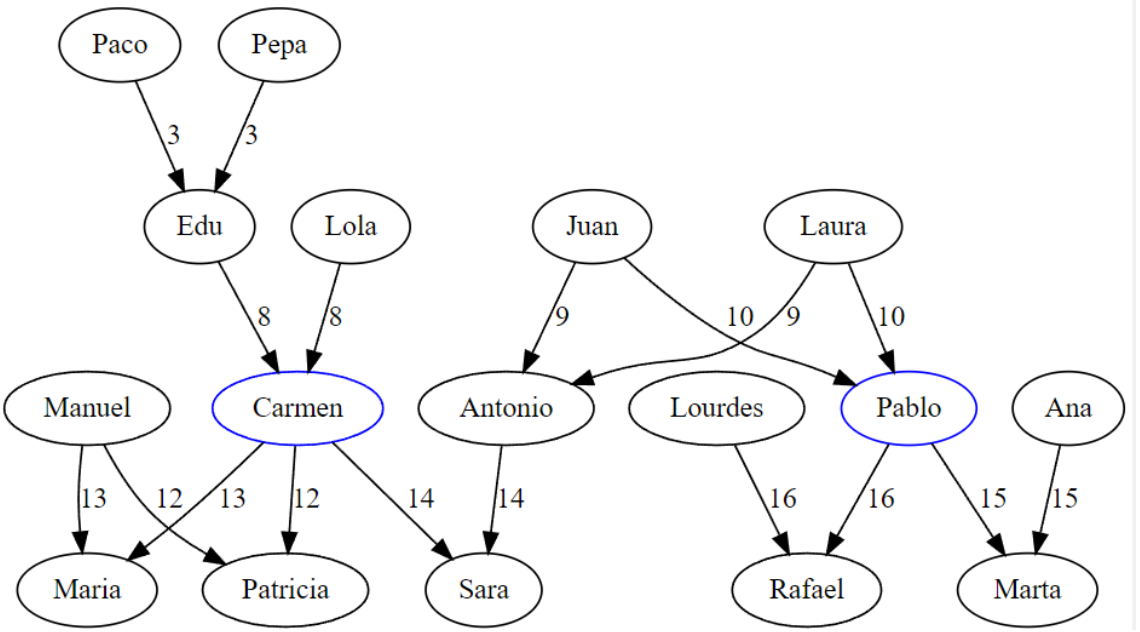
Resultados (consola + grafos)

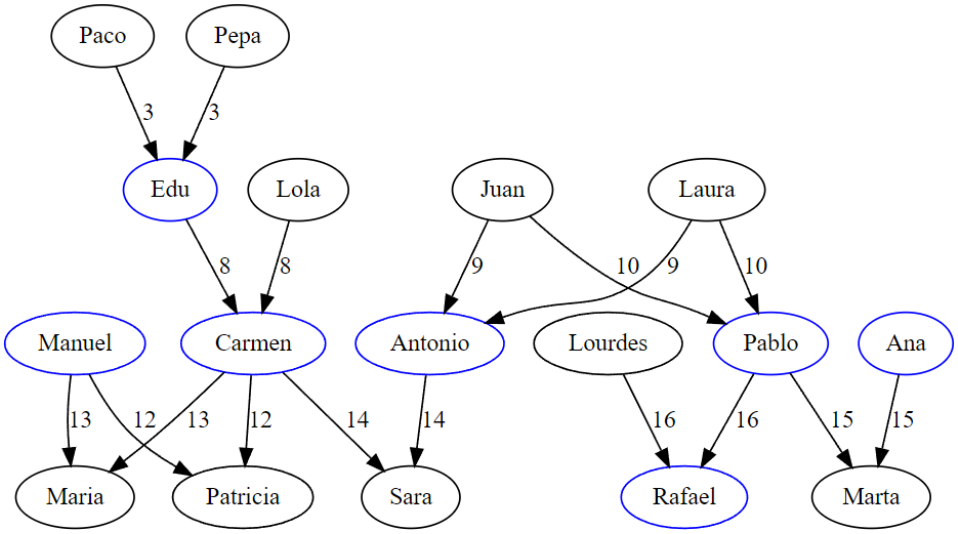
```
APARTADO A:
Se ha generado resultados/ejercicio1/PI3E1A_DatosEntradaApartado A.gv

APARTADO B:
[Personas [id=7, nombre=Manuel, anyo=1982, lugarNacimiento=Madrid], Personas [id=8, nombre=Carmen, anyo=1989, lugarNacimiento=Jaen], Personas [id=3, n
Se ha generado resultados/ejercicio1/PI3E1A_DatosEntradaApartado B.gv

APARTADO C:
Rafael y Carmen son Otros
Rafael y Sara son Primos
Maria y Patricia son Hermanos

APARTADO D:
[Personas [id=8, nombre=Carmen, anyo=1989, lugarNacimiento=Jaen], Personas [id=10, nombre=Pablo, anyo=1991, lugarNacimiento=Sevilla]]
Se ha generado resultados/ejercicio1/PI3E1A_DatosEntradaApartado D.gv

APARTADO E:
El conjunto esta compuesto por 7 personas:
Personas [id=8, nombre=Carmen, anyo=1989, lugarNacimiento=Jaen]
Personas [id=10, nombre=Pablo, anyo=1991, lugarNacimiento=Sevilla]
Personas [id=9, nombre=Antonio, anyo=1989, lugarNacimiento=Jaen]
Personas [id=3, nombre=Edu, anyo=1959, lugarNacimiento=Jerez]
Personas [id=7, nombre=Manuel, anyo=1982, lugarNacimiento=Madrid]
Personas [id=11, nombre=Ana, anyo=1985, lugarNacimiento=Cadiz]
Personas [id=16, nombre=Rafael, anyo=2020, lugarNacimiento=Malaga]
Se ha generado resultados/ejercicio1/PI3E1A_DatosEntradaApartado E.gv

APARTADO A:
Se ha generado resultados/ejercicio1/PI3E1B_DatosEntradaApartado A.gv

APARTADO B:
[Personas [id=11, nombre=Josefina, anyo=1967, lugarNacimiento=Cadiz], Personas [id=9, nombre=Pedro, anyo=1949, lugarNacimiento=Sevilla], Personas [id=
Se ha generado resultados/ejercicio1/PI3E1B_DatosEntradaApartado B.gv

APARTADO C:
Raquel y Laura son Otros
Raquel y Alvaro son Hermanos
Angela y Julia son Primos

APARTADO D:
[Personas [id=11, nombre=Josefina, anyo=1967, lugarNacimiento=Cadiz]]
Se ha generado resultados/ejercicio1/PI3E1B_DatosEntradaApartado D.gv

APARTADO E:
El conjunto esta compuesto por 6 personas:
Personas [id=11, nombre=Josefina, anyo=1967, lugarNacimiento=Cadiz]
Personas [id=4, nombre=Ramon, anyo=1967, lugarNacimiento=Cadiz]
Personas [id=3, nombre=Laura, anyo=1965, lugarNacimiento=Jerez]
Personas [id=9, nombre=Pedro, anyo=1949, lugarNacimiento=Sevilla]
Personas [id=5, nombre=Marcos, anyo=1965, lugarNacimiento=Jerez]
Personas [id=12, nombre=Javier, anyo=1973, lugarNacimiento=Cordoba]
Se ha generado resultados/ejercicio1/PI3E1B_DatosEntradaApartado E.gv
```

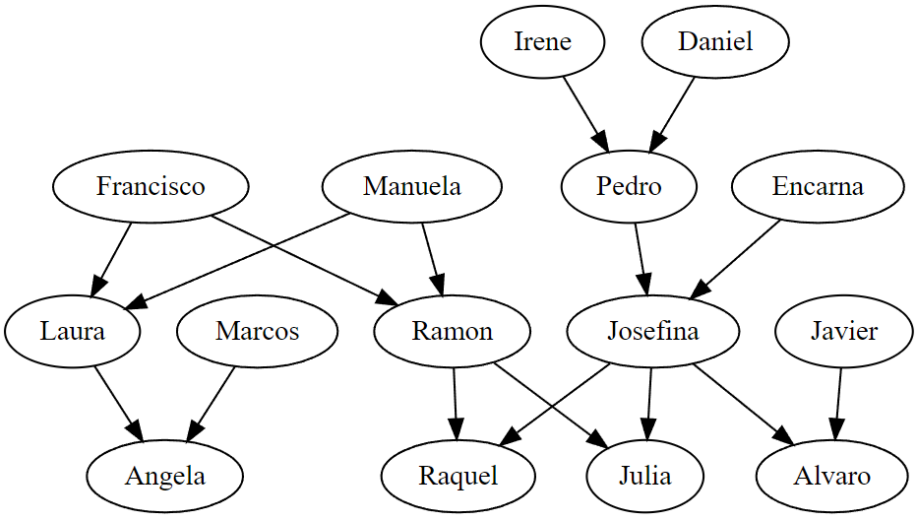


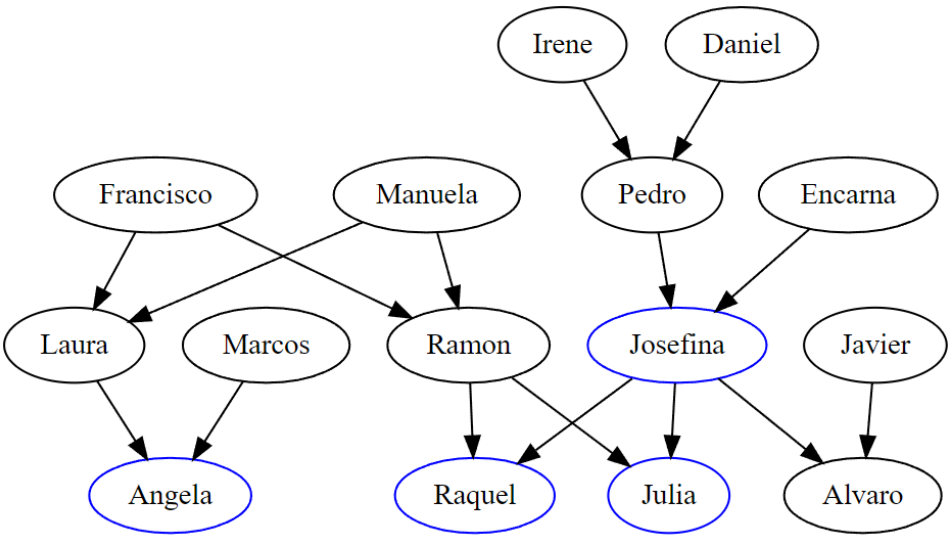

Paco
Pepa
Edu
Lola
Juan
Laura
Manuel
Carmen
Antonio
Ana
Pablo
Lourdes
Maria
Patricia
Sara
Marta
Rafael

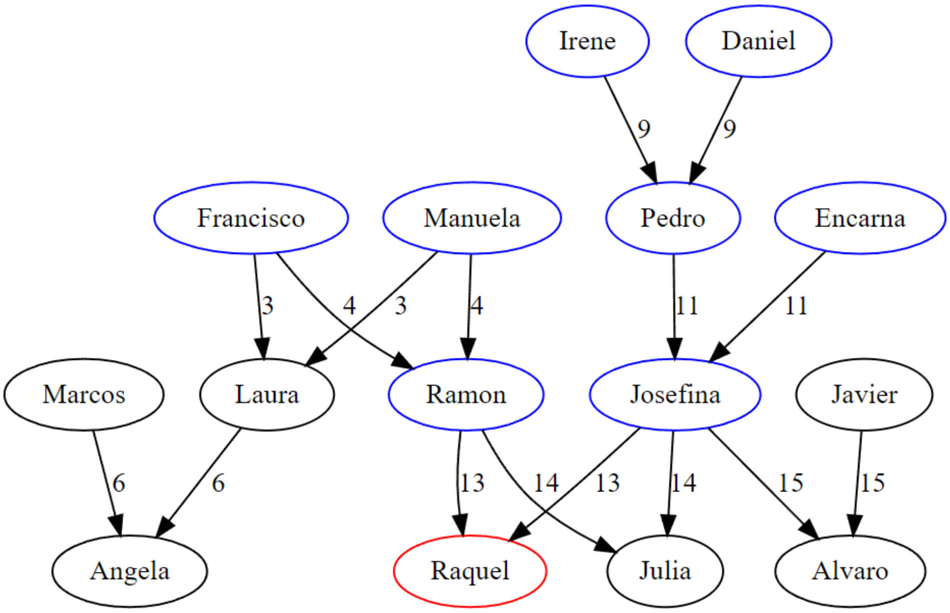

Paco
Pepa
3
3
Edu
Lola
Juan
Laura
8
8
9
10
9
10
Manuel
Carmen
Antonio
Lourdes
Pablo
Ana
13
12
13
12
14
14
16
16
15
15
Maria
Patricia
Sara
Rafael
Marta

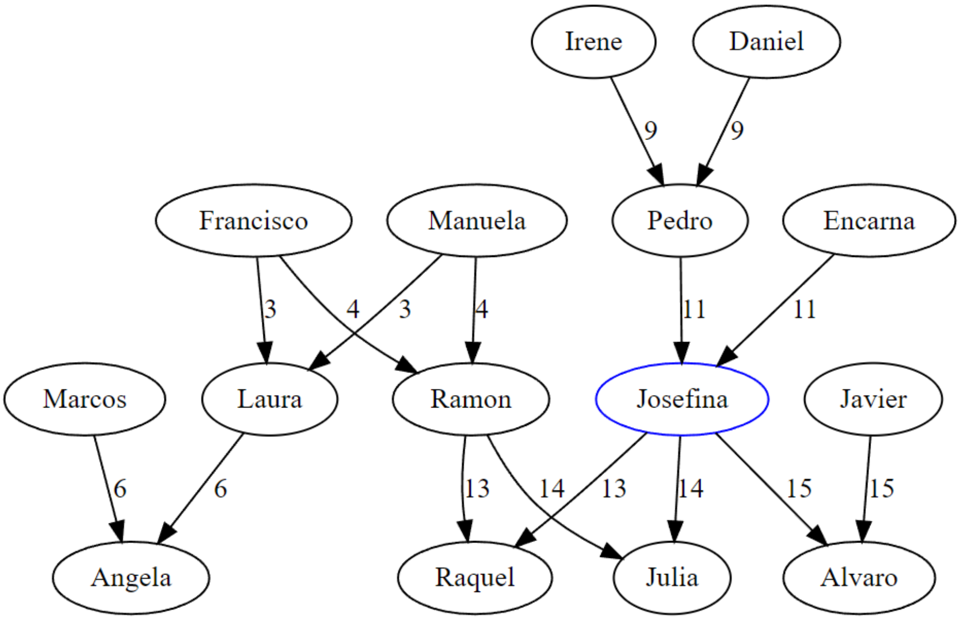

Paco
Pepa
3
3
Edu
Lola
Juan
Laura
8
8
9
10
9
10
Manuel
Carmen
Antonio
Lourdes
Pablo
Ana
13
12
13
12
14
14
16
16
15
15
Maria
Patricia
Sara
Rafael
Marta

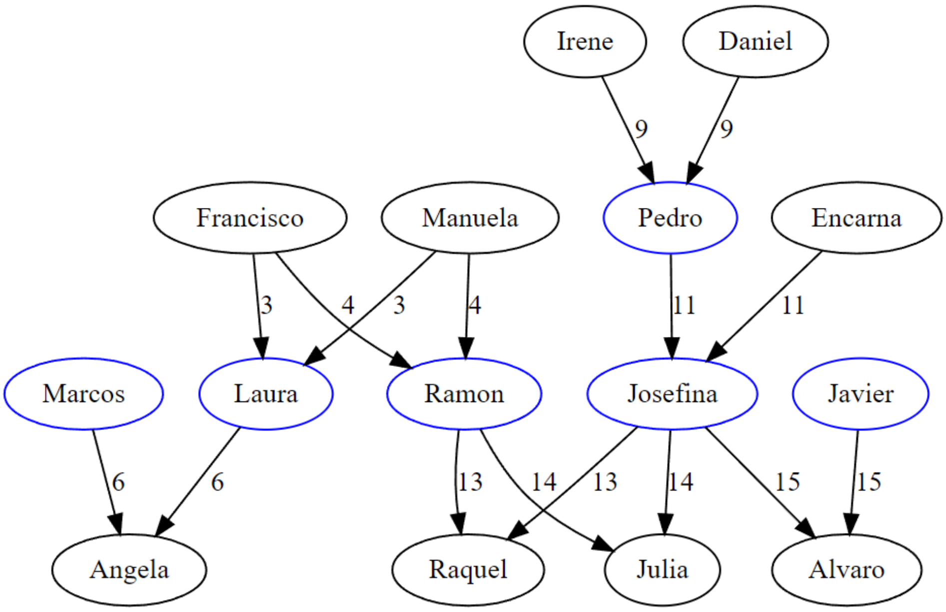

Paco
Pepa
3
3
Edu
Lola
Juan
Laura
8
8
9
10
9
10
Manuel
Carmen
Antonio
Lourdes
Pablo
Ana
13
12
13
12
14
14
16
16
15
15
Maria
Patricia
Sara
Rafael
Marta


Irene
Daniel
Francisco
Manuela
Pedro
Encarna
Laura
Marcos
Ramon
Josefina
Javier
Angela
Raquel
Julia
Alvaro


Irene
Daniel
Francisco
Manuela
Pedro
Encarna
Laura
Marcos
Ramon
Josefina
Javier
Angela
Raquel
Julia
Alvaro


Irene
Daniel
9
9
Francisco
Manuela
Pedro
Encarna
3
4
3
4
11
11
Marcos
Laura
Ramon
Josefina
Javier
6
6
13
14
13
14
15
15
Angela
Raquel
Julia
Alvaro


Irene
Daniel
9
9
Francisco
Manuela
Pedro
Encarna
3
4
3
4
11
11
Marcos
Laura
Ramon
Josefina
Javier
6
6
13
14
13
14
15
15
Angela
Raquel
Julia
Alvaro


Irene
Daniel
9
9
Francisco
Manuela
Pedro
Encarna
3
4
3
4
11
11
Marcos
Laura
Ramon
Josefina
Javier
6
6
13
14
13
14
15
15
Angela
Raquel
Julia
Alvaro

Ejercicio2

Ejercicio2.java

```
package ejercicios;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
import java.util.function.Function;
import java.util.function.Predicate;

import org.jgrapht.Graph;
import org.jgrapht.GraphPath;
import org.jgrapht.Graphs;
import org.jgrapht.alg.connectivity.ConnectivityInspector;

import org.jgrapht.alg.shortestpath.FloydWarshallShortestPaths;
import org.jgrapht.alg.tour.HeldKarpTSP;
import org.jgrapht.graph.SimpleWeightedGraph;

import datos.Ciudad2;

import datos.Trayecto;
import us.lsi.colors.GraphColors;
import us.lsi.colors.GraphColors.Color;
import us.lsi.colors.GraphColors.Style;

import us.lsi.graphs.views.SubGraphView;

public class Ejercicio2 {

	//	Determine cuántos grupos de ciudades hay y cuál es su composición. Dos
	//	ciudades pertenecen al mismo grupo si están relacionadas directamente entre sí o
	//	si existen algunas ciudades intermedias que las relacionan. Muestre el grafo
	//	configurando su apariencia de forma que se coloree cada grupo de un color
	//	diferente.

	private static Color asignaColor(Ciudad2 v, List<Set<Ciudad2>> ls,
ConnectivityInspector<Ciudad2, Trayecto> alg ) {
		Color[] vc = Color.values();
		Set<Ciudad2> s = alg.connectedSetOf(v);
		return vc [ls.indexOf(s)];
	}

	public static List<Set<Ciudad2>> apartadoA(Graph<Ciudad2, Trayecto> g, String file,
String nombreVista) {
		ConnectivityInspector<Ciudad2, Trayecto> p = new ConnectivityInspector<>(g);
		List<Set<Ciudad2>> ls = p.connectedSets();

		String fileRes = "resultados/ejercicio2/" + file + nombreVista +  ".gv";
		GraphColors.toDot(g, fileRes,
				v->v.id(), e->"",
				v-> GraphColors.color(asignaColor(v, ls, p)),
				e-> GraphColors.color(asignaColor(g.getEdgeSource(e), ls, p)));

		System.out.println("Hay " + ls.size() + " componentes conexas");
		System.out.println("Se ha generado " + fileRes);
		return ls;
	}
```

```
	//	Determine cuál es el grupo de ciudades a visitar si se deben elegir las ciudades
	//	conectadas entre sí que maximice la suma total de las puntuaciones. Muestre el
	//	grafo configurando su apariencia de forma que se resalten dichas ciudades.

	public static void apartadoB(Graph<Ciudad2, Trayecto> g, String file, String
nombreVista) {
		//obtengo componentes conexas
		List<Set<Ciudad2>> ls = apartadoA(g, file, nombreVista);
		//funcion de sumatorio de puntuaciones
		Function<Set<Ciudad2>, Integer> algoritmo = elem ->
elem.stream().map(Ciudad2::puntuacion).reduce(0, (a, b) -> a + b);

		//calculo sumas para cada componente conexa
		Integer puntuacion = ls.stream().map(algoritmo).mapToInt(x-
>x).max().getAsInt();//calculo la puntuación maxima para comprobar
		Set<Ciudad2> res = ls.stream().reduce((a,b)-> algoritmo.apply(a) >
algoritmo.apply(b)? a:b).get();//me quedo con el conjunto de vertices con mayor puntuacion
		System.out.println("Los vertices que tienen la suma maxima de puntuaciones son: "
+ res + " y la suma de sus puntuaciones es: " + puntuacion);//imprimo para comprobar
		Graph<Ciudad2, Trayecto> g2 = SubGraphView.of(g, res);//creo una subvista con el
conjunto de mayor puntuacion

		String fileRes = "resultados/ejercicio2/" + file + nombreVista +  ".gv";
		GraphColors.toDot(g, fileRes,
				v->v.id(), e->"",
				v-> GraphColors.colorIf(Color.blue, g2.vertexSet().contains(v)),
				e-> GraphColors.colorIf(Color.blue, g2.edgeSet().contains(e)));


		System.out.println("Se ha generado " + fileRes);
	}


	//	Determine cuál es el grupo de ciudades a visitar si se deben elegir las ciudades
	//	conectadas entre sí que den lugar al camino cerrado de menor precio que pase por
	//	todas ellas. Muestre el grafo configurando su apariencia de forma que se resalte
	//	dicho camino.

	public static void apartadoC(SimpleWeightedGraph<Ciudad2, Trayecto> g, String file,
String nombreVista) {
		//obtengo componentes conexas
		List<Set<Ciudad2>> ls = apartadoA(g, file, nombreVista);
		//algoritmo hamilton problema del viajante
		HeldKarpTSP<Ciudad2, Trayecto> alg = new HeldKarpTSP<>();
		List<GraphPath<Ciudad2, Trayecto>> listaCaminos = new ArrayList<>(); //lista donde
almaceno los caminos
		//calculo caminos para cada componente conexa
		ls.forEach(y -> {
			Graph<Ciudad2, Trayecto> g2 = SubGraphView.of(g, y);
			GraphPath<Ciudad2, Trayecto> r = alg.getTour(g2);
			listaCaminos.add(r);
		});

		GraphPath<Ciudad2, Trayecto> res = //me quedo con el camino con menor longitud
				listaCaminos.stream()
				.reduce((a,b)-> a.getWeight() < b.getWeight()? a:b)
				.get();

		System.out.println("Los vertices del camino de menor precio son: " +
res.getVertexList());
```

```
		System.out.println("El coste es: " + res.getWeight());

		Set<Ciudad2> verticesCamino = new HashSet<Ciudad2>(res.getVertexList());
		Set<Trayecto> aristasCamino = new HashSet<Trayecto>(res.getEdgeList());

		String fileRes = "resultados/ejercicio2/" + file + nombreVista +  ".gv";
		GraphColors.toDot(g, fileRes,
				v->v.id(), e->e.precio().toString(),
				v-> GraphColors.colorIf(Color.blue, verticesCamino.contains(v)),
				e-> GraphColors.colorIf(Color.blue, aristasCamino.contains(e)));

		System.out.println("Se ha generado " + fileRes);

	}

	//	De cada grupo de ciudades, determinar cuáles son las 2 ciudades (no conectadas
	//	directamente entre sí) entre las que se puede viajar en un menor tiempo. Muestre
	//	el grafo configurando su apariencia de forma que se resalten las ciudades y el
	//	camino entre ellas.

	public static void apartadoD(SimpleWeightedGraph<Ciudad2, Trayecto> g, String file,
String nombreVista) {

		ConnectivityInspector<Ciudad2, Trayecto> p = new ConnectivityInspector<>(g);
		List<Set<Ciudad2>> ls = p.connectedSets();
		List<GraphPath<Ciudad2, Trayecto>> listaCaminos = new ArrayList<>();//lista donde
almaceno el camino más corto para cada componente
		ls.forEach(x -> {		//itero en cada componente conexa
			Graph<Ciudad2, Trayecto> g2 = SubGraphView.of(g, x);

			List<Ciudad2> vertices = g2.vertexSet().stream().toList();

			FloydWarshallShortestPaths<Ciudad2, Trayecto> a = new
FloydWarshallShortestPaths<>(g2); //algoritmo camino minimo
			Set<GraphPath<Ciudad2, Trayecto>> s = new HashSet<>();

			for(Ciudad2 v : vertices) { //para cada vertice
				List<Ciudad2> vertices2 = vertices;
				vertices2 = vertices2.stream() //guardo en una lista el conjunto de
vertices del grafo borrando los adyacentes del vertice donde estoy
						.filter(y -> !Graphs.successorListOf(g2, v).contains(y)
&& !Graphs.predecessorListOf(g2, v).contains(y))
						.toList();

				for(Ciudad2 v2 : vertices2) {
					if(a.getPath(v,v2).getWeight()!=0) s.add(a.getPath(v,v2));
//calculo los caminos minimos entre mi vertice y cada uno de los que no son adyacentes a el
				}
			}

			GraphPath<Ciudad2, Trayecto> caminoMinimo = s.stream().reduce((a3, b) ->
a3.getWeight() < b.getWeight()? a3: b).get(); //voy comparando peso y me quedo con el camino
con menor peso
			listaCaminos.add(caminoMinimo); //lo añado a la lista de caminos (este ya
sí es el camino de menor peso en esta componente conexa
			System.out.println("El camino que realiza es: " + caminoMinimo + " y su
peso es: " + caminoMinimo.getWeight());

			Predicate<Ciudad2> pv1 = c-> caminoMinimo.getVertexList().contains(c);
			Predicate<Trayecto> pa1 = e -> caminoMinimo.getEdgeList().contains(e);
```

```
                    Graph<Ciudad2, Trayecto> gRes = SubGraphView.of(g, pv1, pa1);
                    int componente = ls.indexOf(x) + 1;
                    String fileRes = "resultados/ejercicio2/" + file + nombreVista + "-
componente-" + componente +  ".gv";
                    GraphColors.toDot(g, fileRes,
                                v->v.id(), e->e.precio().toString(),
                                v-> GraphColors.styleIf(Style.bold,
gRes.vertexSet().contains(v)),
                                e-> GraphColors.styleIf(Style.bold,
gRes.edgeSet().contains(e)));

                    System.out.println("Se ha generado " + fileRes);

             });

      }

}
```

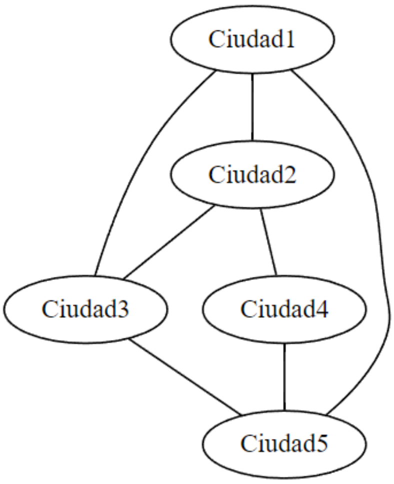
TestEjercicio2.java

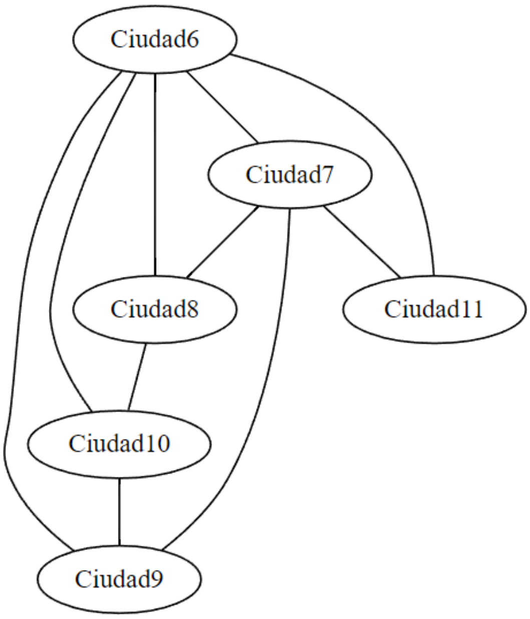
```
package tests;

import org.jgrapht.Graph;
import org.jgrapht.graph.SimpleWeightedGraph;

import datos.Ciudad2;
import datos.Trayecto;
import ejercicios.Ejercicio2;
import us.lsi.colors.GraphColors;
import us.lsi.colors.GraphColors.Color;
import us.lsi.graphs.Graphs2;
import us.lsi.graphs.GraphsReader;

public class TestEjercicio2 {

      public static void main(String[] args) {
            testEjercicio2("PI3E2_DatosEntrada");

      }

      private static void testEjercicio2(String file) {
            Graph<Ciudad2, Trayecto> g = GraphsReader.newGraph("ficheros/" + file + ".txt",
Ciudad2::ofFormat, Trayecto::ofFormat,
                        Graphs2::simpleWeightedGraph);

            SimpleWeightedGraph<Ciudad2, Trayecto> g2 = GraphsReader.newGraph("ficheros/" +
file + ".txt",
                        Ciudad2::ofFormat,
                        Trayecto::ofFormat,
                        Graphs2::simpleWeightedGraph,
                        Trayecto::duracion
                        );

            SimpleWeightedGraph<Ciudad2, Trayecto> g3 = GraphsReader.newGraph("ficheros/" +
file + ".txt",
                        Ciudad2::ofFormat,
                        Trayecto::ofFormat,
                        Graphs2::simpleWeightedGraph,
                        Trayecto::precio
```

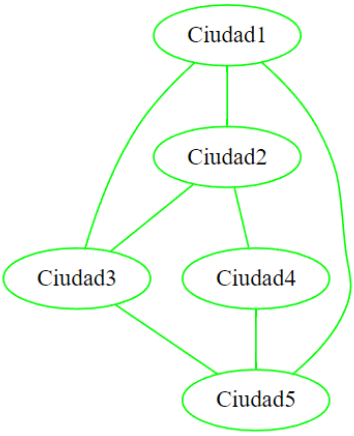
```
                        );


                String fileRes = "resultados/ejercicio2/"+ file + ".gv";
                GraphColors.toDot(g, fileRes,
                                v -> v.id(), e -> "",
                                v -> GraphColors.color(Color.black), e ->
GraphColors.color(Color.black));


                //Apartado A
                System.out.println("APARTADO A:");
                System.out.println(Ejercicio2.apartadoA(g, file, "Apartado A"));
                System.out.println(" ");

                //Apartado B
                System.out.println("APARTADO B:");
                Ejercicio2.apartadoB(g, file, "Apartado B");
                System.out.println(" ");

                //Apartado C
                System.out.println("APARTADO C:");
                Ejercicio2.apartadoC(g3, file, "Apartado C");
                System.out.println(" ");

                //Apartado D
                System.out.println("APARTADO D:");
                Ejercicio2.apartadoD(g2, file, "Apartado D");
                System.out.println(" ");

        }

}
```

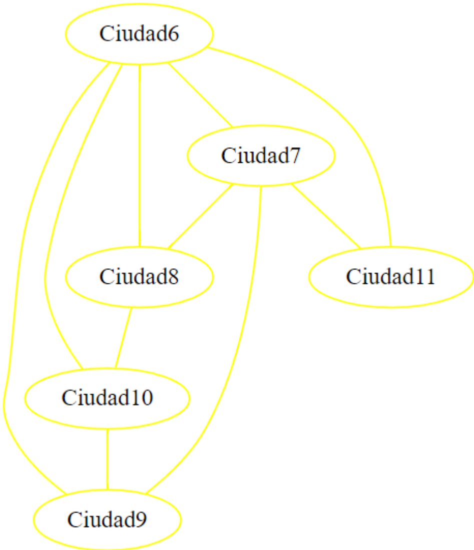
Resultados (consola + grafos)

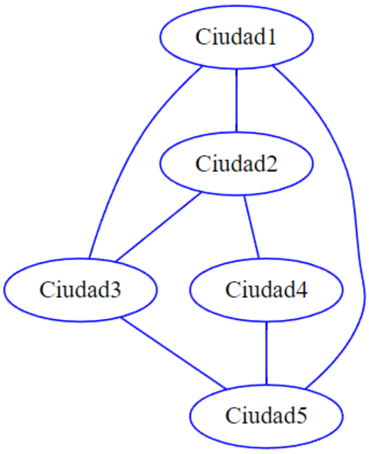
```
APARTADO A:
Hay 2 componentes conexas
Se ha generado resultados/ejercicio2/PI3E2_DatosEntradaApartado A.gv
[[Ciudad5, Ciudad2, Ciudad4, Ciudad3, Ciudad1], [Ciudad8, Ciudad11, Ciudad10, Ciudad6, Ciudad7, Ciudad9]]

APARTADO B:
Hay 2 componentes conexas
Se ha generado resultados/ejercicio2/PI3E2_DatosEntradaApartado B.gv
Los vertices que tienen la suma maxima de puntuaciones son: [Ciudad5, Ciudad2, Ciudad4, Ciudad3, Ciudad1] y la suma de sus puntuaciones es: 115
Se ha generado resultados/ejercicio2/PI3E2_DatosEntradaApartado B.gv

APARTADO C:
Hay 2 componentes conexas
Se ha generado resultados/ejercicio2/PI3E2_DatosEntradaApartado C.gv
Los vertices del camino de menor precio son: [Ciudad8, Ciudad10, Ciudad9, Ciudad7, Ciudad11, Ciudad6, Ciudad8]
El coste es: 60.0
Se ha generado resultados/ejercicio2/PI3E2_DatosEntradaApartado C.gv

APARTADO D:
El camino que realiza es: [Trayecto[origen=Ciudad2, destino=Ciudad3, precio=20.0, duracion=15.0], Trayecto[origen=Ciudad2, destino=Ciudad4, precio=20.0, duracion=25.0]] y su peso es: 40.0
Se ha generado resultados/ejercicio2/PI3E2_DatosEntradaApartado D-componente-1.gv
El camino que realiza es: [Trayecto[origen=Ciudad7, destino=Ciudad9, precio=5.0, duracion=25.0], Trayecto[origen=Ciudad7, destino=Ciudad8, precio=4.0, duracion=10.0]] y su peso es: 35.0
Se ha generado resultados/ejercicio2/PI3E2_DatosEntradaApartado D-componente-2.gv
```

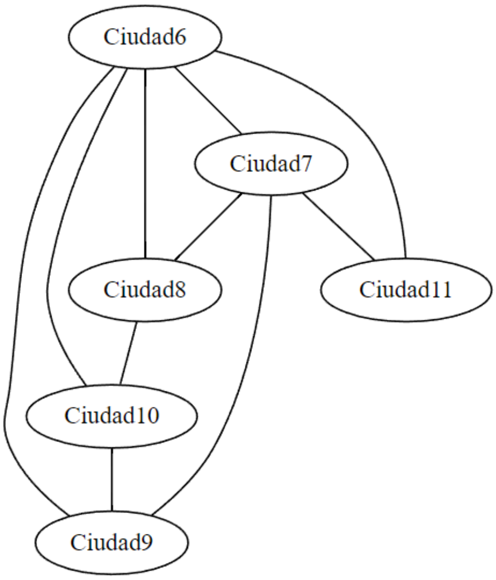

Ciudad1
Ciudad2
Ciudad3
Ciudad4
Ciudad5

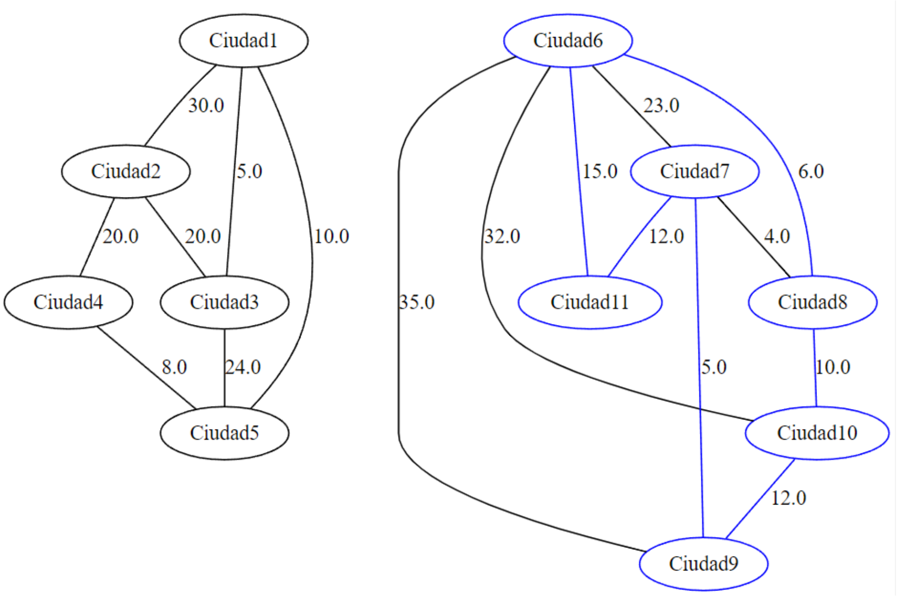

Ciudad6
Ciudad7
Ciudad8
Ciudad11
Ciudad10
Ciudad9

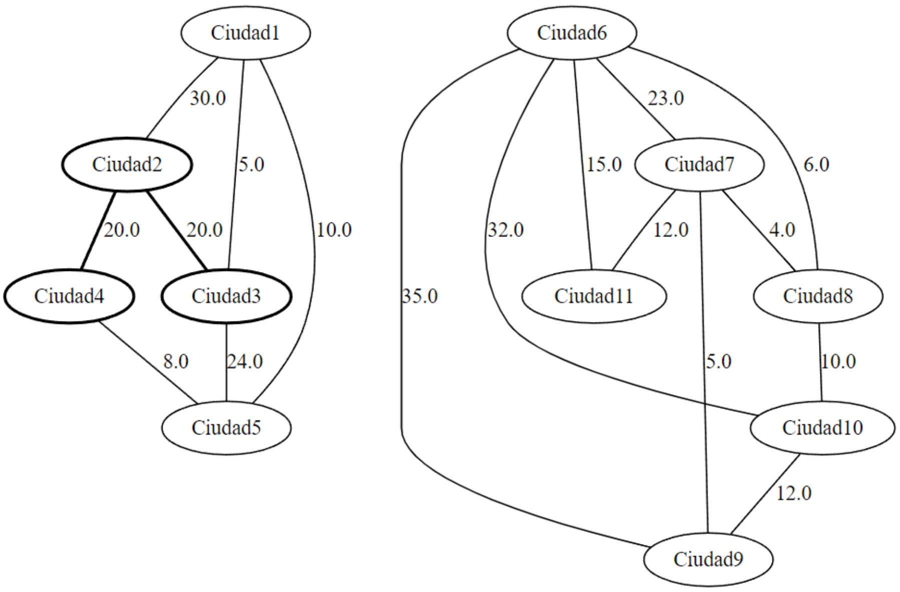

Ciudad1
Ciudad2
Ciudad3
Ciudad4
Ciudad5

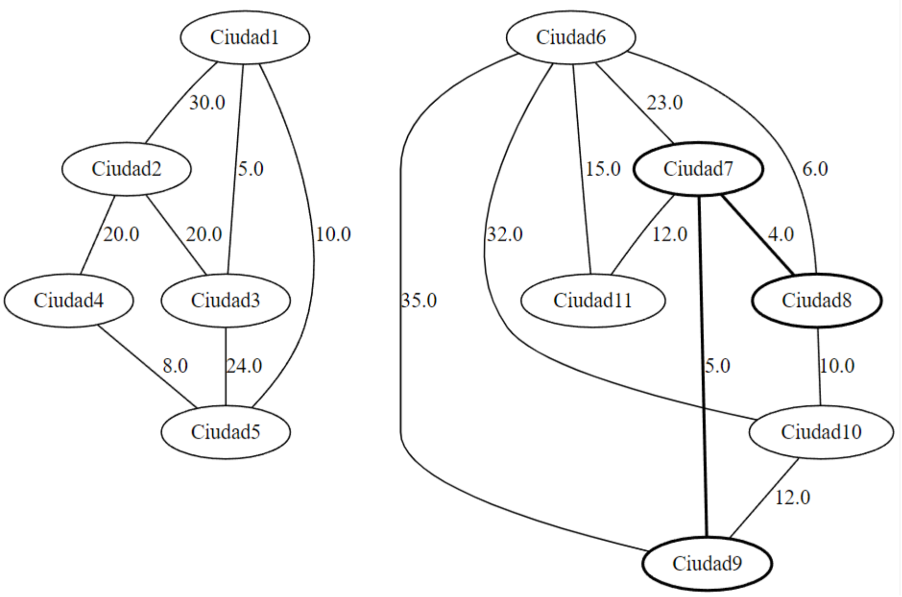

Ciudad6
Ciudad7
Ciudad8
Ciudad11
Ciudad10
Ciudad9


Ciudad1
Ciudad2
Ciudad3
Ciudad4
Ciudad5


Ciudad6
Ciudad7
Ciudad8
Ciudad11
Ciudad10
Ciudad9


Ciudad1
30.0
5.0
10.0
Ciudad2
20.0
20.0
Ciudad4
Ciudad3
8.0
24.0
Ciudad5
Ciudad6
23.0
15.0
Ciudad7
6.0
32.0
12.0
4.0
35.0
Ciudad11
Ciudad8
5.0
10.0
Ciudad10
12.0
Ciudad9


Ciudad1
30.0
5.0
10.0
Ciudad2
20.0
20.0
Ciudad4
Ciudad3
8.0
24.0
Ciudad5
Ciudad6
23.0
15.0
Ciudad7
6.0
32.0
12.0
4.0
35.0
Ciudad11
Ciudad8
5.0
10.0
Ciudad10
12.0
Ciudad9


Ciudad1
30.0
5.0
10.0
Ciudad2
20.0
20.0
Ciudad4
Ciudad3
8.0
24.0
Ciudad5
Ciudad6
23.0
15.0
Ciudad7
6.0
32.0
12.0
4.0
35.0
Ciudad11
Ciudad8
5.0
10.0
Ciudad10
12.0
Ciudad9

Ejercicio3

Ejercicio3.java

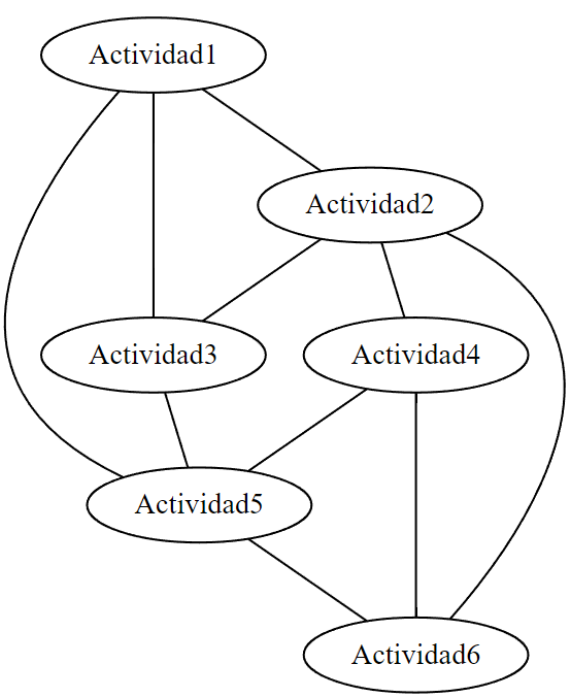
```
package ejercicios;

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

import org.jgrapht.Graph;
import org.jgrapht.alg.color.GreedyColoring;
import org.jgrapht.alg.interfaces.VertexColoringAlgorithm.Coloring;
import org.jgrapht.graph.DefaultEdge;

import us.lsi.colors.GraphColors;
import us.lsi.colors.GraphColors.Style;

public class Ejercicio3 {
	public static void todosLosApartados(Graph<String, DefaultEdge> g, String file) {

		GreedyColoring<String, DefaultEdge> alg = new GreedyColoring<>(g);
		Coloring<String> coloreado = alg.getColoring();
		System.out.println("Numero de franjas horarias necesarias: " +
coloreado.getNumberColors());
		System.out.println("Actividades para impartirse en paralelo por franja horaria:
");

		List<Set<String>> comp = coloreado.getColorClasses();
		for(int i = 0; i < comp.size(); i++) {
			System.out.println("Franja numero " + (i+1) + ": " + comp.get(i));
		}


		Map<String, Integer> map = coloreado.getColors();
		String fileRes = "resultados/ejercicio3/" + file  + "B.gv";
		GraphColors.toDot(g, fileRes,
				v -> v,
				e -> "",
				v -> GraphColors.color(map.get(v)),
				e -> GraphColors.style(Style.bold));

		System.out.println("Se ha generado " + fileRes);
	}
}
```

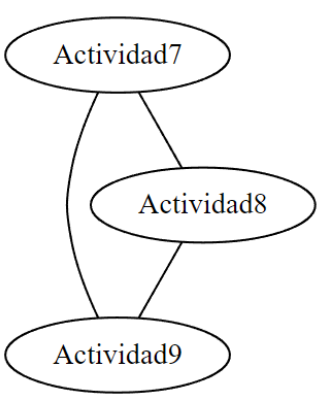
TestEjercicio3.java

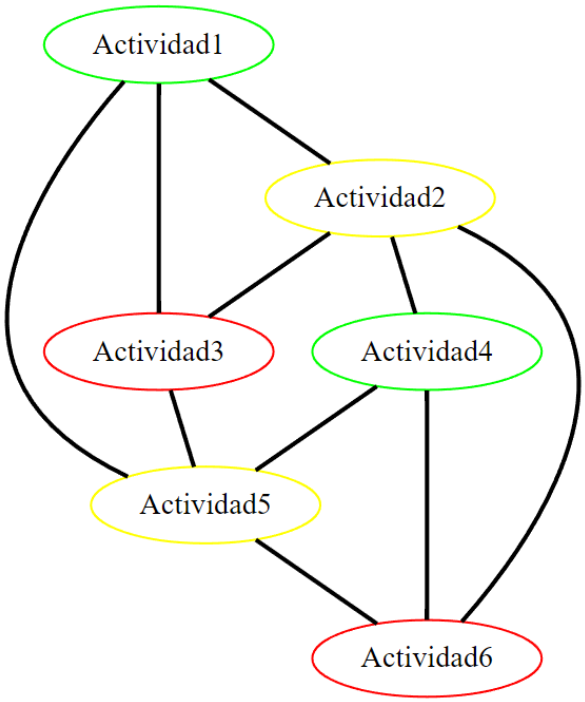
```
package tests;

import org.jgrapht.Graph;
import org.jgrapht.graph.DefaultEdge;

import ejemplos.Ejemplo3;
import ejercicios.Ejercicio3;
import us.lsi.colors.GraphColors;
import us.lsi.colors.GraphColors.Color;
import us.lsi.common.Files2;
import us.lsi.graphs.Graphs2;

public class TestEjercicio3 {

	public static void main(String[] args) {
		testEjercicio3("PI3E3A_DatosEntrada");
		testEjercicio3("PI3E3B_DatosEntrada");
	}

	public static void testEjercicio3(String file) {

	Graph<String, DefaultEdge> g =
		Graphs2.simpleGraph(String::new, DefaultEdge::new, false);


	Files2.streamFromFile("ficheros/" + file + ".txt").forEach(linea -> {
		String[] v1 = linea.trim().split(":");
		String[] v2 = v1[1].trim().split(", ");

		for (int i = 0; i < v2.length - 1; i++) {
			if (!g.containsVertex(v2[i])) g.addVertex(v2[i]);
			for (int j = i + 1; j < v2.length; j++) {
				if (!g.containsVertex(v2[j])) g.addVertex(v2[j]);
				g.addEdge(v2[i], v2[j]);
			}
		}

	});

	//Generar el archivo gv
	String fileRes = "resultados/ejercicio3/" + file + ".gv";
	GraphColors.toDot(g, fileRes,
		v -> v, e-> "",
		v-> GraphColors.color(Color.black),
		e-> GraphColors.color(Color.black)
		);

	Ejercicio3.todosLosApartados(g, file);

}

}
```

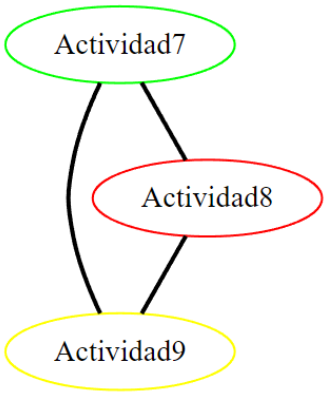
Resultados (consola + grafos)

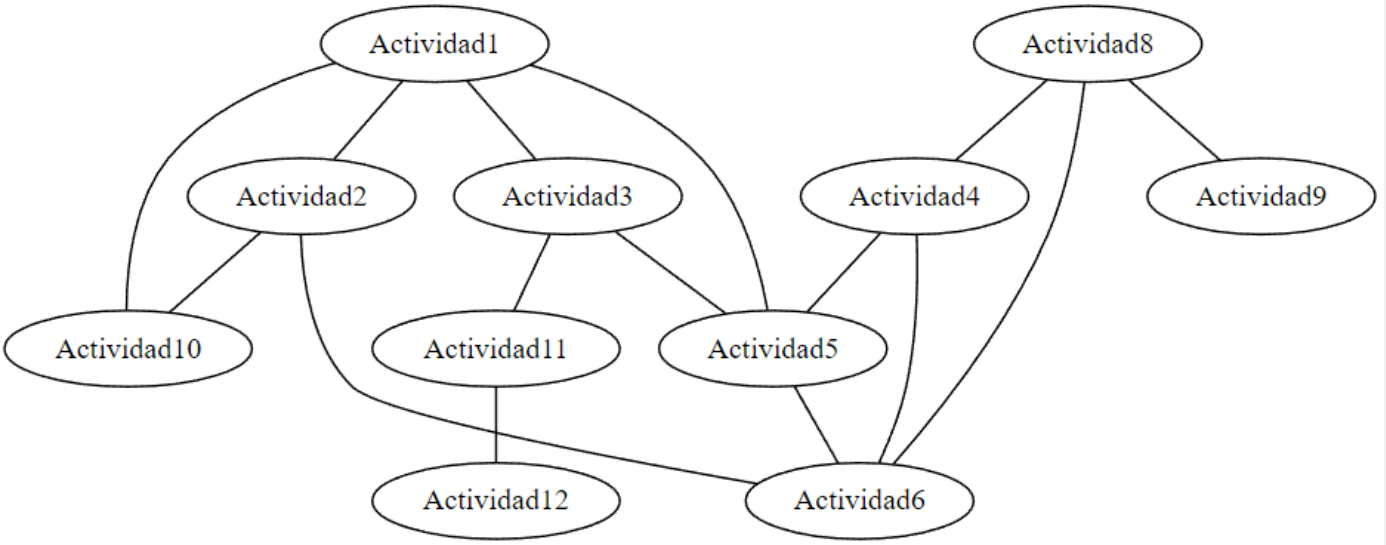
```
Numero de franjas horarias necesarias: 3
Actividades para impartirse en paralelo por franja horaria:
Franja numero 1: [Actividad1, Actividad4, Actividad7]
Franja numero 2: [Actividad2, Actividad9, Actividad5]
Franja numero 3: [Actividad3, Actividad6, Actividad8]
Se ha generado resultados/ejercicio3/PI3E3A_DatosEntradaB.gv
Numero de franjas horarias necesarias: 4
Actividades para impartirse en paralelo por franja horaria:
Franja numero 1: [Actividad1, Actividad11, Actividad8]
Franja numero 2: [Actividad2, Actividad4, Actividad3, Actividad9, Actividad12]
Franja numero 3: [Actividad10, Actividad5]
Franja numero 4: [Actividad6]
Se ha generado resultados/ejercicio3/PI3E3B_DatosEntradaB.gv
```

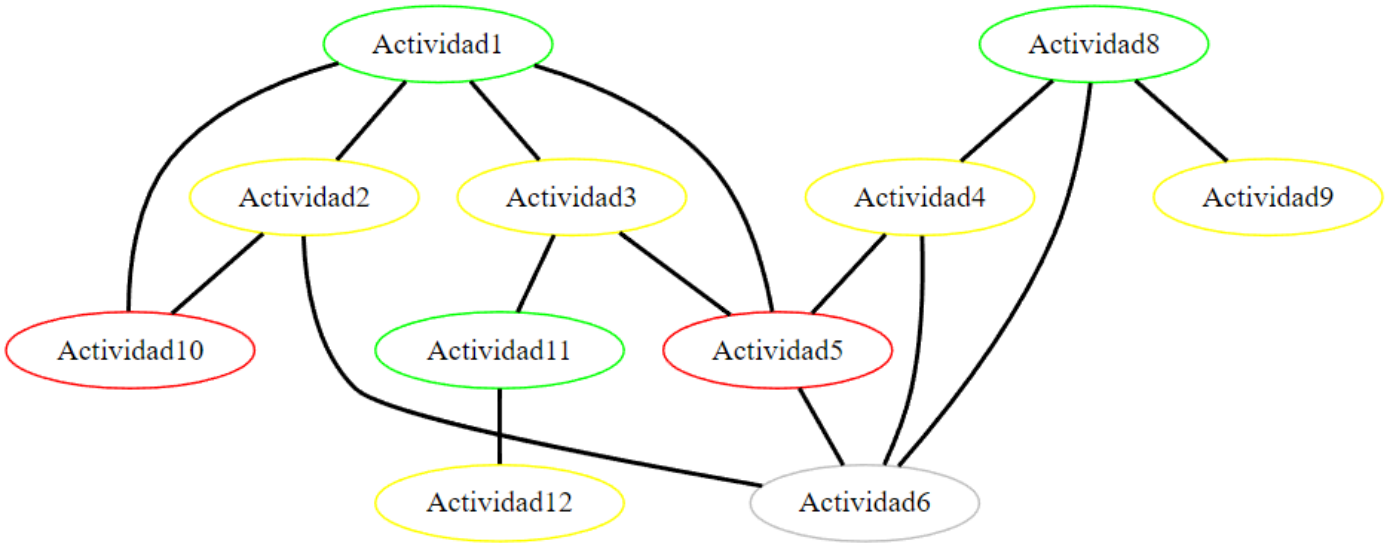









Actividad1
Actividad8
Actividad2
Actividad3
Actividad4
Actividad9
Actividad10
Actividad11
Actividad5
Actividad12
Actividad6


Actividad1
Actividad8
Actividad2
Actividad3
Actividad4
Actividad9
Actividad10
Actividad11
Actividad5
Actividad12
Actividad6