MEMORIA

PRACTICA INDIVIDUAL 3

Jaime Linares Barrera

2º Ingeniería Informática - Ingeniería del Software, Grupo 4

1. EJERCICIO 1

1.1. CÓDIGO EJERCICIO

```java
package ejercicios;

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

import org.jgrapht.Graph;
import org.jgrapht.Graphs;
import org.jgrapht.alg.vertexcover.GreedyVCImpl;

import tipos.Familiar;
import tipos.Persona;
import tipos.Relacion;
import us.lsi.colors.GraphColors;
import us.lsi.colors.GraphColors.Color;
import us.lsi.common.List2;
import us.lsi.graphs.views.SubGraphView;

public class Ejercicio1 {

    private static String carpetaResultados = "resultados/ejercicio1/";

    // APARTADO A
    public static Set<Persona> fEjercicio1A(Graph<Persona,Relacion> gf, String fichero) {
        Set<Persona> res = new HashSet<>();
        Set<Persona> personas = gf.vertexSet();

        List<Persona> padres = new ArrayList<>();
        for(Persona persona: personas) {
            padres = Graphs.predecessorListOf(gf, persona);
            if(padres.size()==2 && padres.get(0).ciudad().equals(padres.get(1).ciudad()) &&
                    padres.get(0).anyo().equals(padres.get(1).anyo())) {
                res.add(persona);
            }
        }

        Graph<Persona,Relacion> sgf = SubGraphView.of(gf,
                p -> res.contains(p),
                r -> res.contains(gf.getEdgeSource(r)) && res.contains(gf.getEdgeTarget(r)));

        GraphColors.toDot(gf,
                carpetaResultados + fichero + "_ApartadoA.gv",
                p -> p.nombre(),
                r -> "",
                p -> GraphColors.colorIf(Color.blue, sgf.containsVertex(p)),
                r -> GraphColors.colorIf(Color.blue, sgf.containsEdge(r)));

        System.out.println("- Fichero procesado en " + fichero + "_ApartadoA.gv en la carpeta " + carpetaResultados);
        return res;
    }


    // APARTADO B
    public static Set<Persona> fEjercicio1B(Graph<Persona,Relacion> gf, Persona persona, String fichero) {
        Set<Persona> res = new HashSet<>();

        res = obtieneAncestros(gf, persona, res);

        representaGrafo(gf, persona, fichero, res);

        return res;
    }

    private static Set<Persona> obtieneAncestros(Graph<Persona, Relacion> gf, Persona p, Set<Persona> ac) {
        List<Persona> padres = Graphs.predecessorListOf(gf, p);
        for(Persona padre: padres) {
            ac.add(padre);
            ac = obtieneAncestros(gf, padre, ac);
        }
        return ac;
    }

    private static void representaGrafo(Graph<Persona,Relacion> gf, Persona persona, String fichero,
            Set<Persona> res) {
        GraphColors.toDot(gf,
                carpetaResultados + fichero + "_ApartadoB.gv",
                p -> p.nombre(),
                r -> "",
                p -> GraphColors.colorIf(List2.of(p.equals(persona), res.contains(p)),
                        List2.of(Color.red, Color.blue)),
                r -> GraphColors.color(Color.black));

        System.out.println("- Fichero procesado en " + fichero + "_ApartadoB.gv en la carpeta " + carpetaResultados);
    }
```

```java
    // APARTADO C
    public static Familiar fEjercicio1C(Graph<Persona,Relacion> gf, Persona p1, Persona p2) {
        Familiar res = Familiar.OTROS;
        List<Persona> padresp1 = Graphs.predecessorListOf(gf, p1);
        List<Persona> padresp2 = Graphs.predecessorListOf(gf, p2);
        List<Persona> abuelosp1 = new ArrayList<>();
        List<Persona> abuelosp2 = new ArrayList<>();

        if(!Collections.disjoint(padresp1, padresp2)) {
            res = Familiar.HERMANOS;
        } else {
            // Metemos abuelos p1
            for(Persona padrep1: padresp1) {
                abuelosp1.addAll(Graphs.predecessorListOf(gf, padrep1));
            }
            // Metemos abuelos p2
            for (Persona padrep2: padresp2) {
                abuelosp2.addAll(Graphs.predecessorListOf(gf, padrep2));
            }
            if(!Collections.disjoint(abuelosp1, abuelosp2)) {
                res = Familiar.PRIMOS;
            }
        }

        return res;
    }
```

```java
    // APARTADO D
    public static Set<Persona> fEjercicio1D(Graph<Persona,Relacion> gf, String fichero) {
        Set<Persona> res = new HashSet<>();
        Set<Persona> personas = gf.vertexSet();

        for(Persona persona: personas) {
            List<Persona> hijos = Graphs.successorListOf(gf, persona);
            if(!hijos.isEmpty()) {
                Set<Persona> padresDistintos = new HashSet<>();
                for(Persona hijo: hijos) {
                    padresDistintos.addAll(Graphs.predecessorListOf(gf, hijo));
                }
                if(padresDistintos.size() > 2) {
                    res.add(persona);
                }
            }
        }

        GraphColors.toDot(gf,
                carpetaResultados + fichero + "_ApartadoD.gv",
                p -> p.nombre(),
                r -> "",
                p -> GraphColors.colorIf(Color.blue, res.contains(p)),
                r -> GraphColors.color(Color.black));

        System.out.println("- Fichero procesado en " + fichero + "_ApartadoD.gv en la carpeta " + carpetaResultados);
        return res;
    }
```

```java
    // APARTADO E
    public static Set<Persona> fEjercicio1E(Graph<Persona,Relacion> gf, String fichero) {
        Graph<Persona,Relacion> g = Graphs.undirectedGraph(gf);

        var alg = new GreedyVCImpl<>(g);
        Set<Persona> res = alg.getVertexCover();

        GraphColors.toDot(g,
                carpetaResultados + fichero + "_ApartadoE.gv",
                p -> p.nombre(),
                r -> "",
                p -> GraphColors.colorIf(Color.blue, res.contains(p)),
                r -> GraphColors.color(Color.black));

        System.out.println("- Fichero procesado en " + fichero + "_ApartadoE.gv en la carpeta " + carpetaResultados);
        return res;
    }

}
```

```java
package tipos;

public enum Familiar {
    HERMANOS, PRIMOS, OTROS
}
```

```java
package tipos;

public record Persona(Integer id, String nombre, Integer anyo, String ciudad) {

    public static Persona of(Integer id, String nombre, Integer anyo, String ciudad) {
        return new Persona(id, nombre, anyo, ciudad);
    }

    public static Persona ofFormat(String[] partes) {
        Integer id = Integer.valueOf(partes[0].trim());
        String nombre = partes[1].trim();
        Integer anyo = Integer.valueOf(partes[2].trim());
        String ciudad = partes[3].trim();
        return Persona.of(id, nombre, anyo, ciudad);
    }

    @Override
    public String toString() {
        return this.nombre + " " + this.anyo + " " + this.ciudad;
    }

}
```

```java
package tipos;

public record Relacion(Integer id) {

    private static Integer numId = 0;

    public static Relacion of() {
        Integer id = numId;
        numId += 1;
        return new Relacion(id);
    }

    public static Relacion ofFormat(String[] partes) {
        return Relacion.of();
    }

}
```

1.2. CÓDIGO TEST

```java
package tests;

import org.jgrapht.Graph;

import ejercicios.Ejercicio1;
import tipos.Persona;
import tipos.Relacion;
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) {
        System.out.println("* TEST EJERCICIO 1 *");

        System.out.println("\n---------- DATOS DE ENTRADA A ----------");
        testsEjercicio1("PI3E1A_DatosEntrada");

        System.out.println("\n---------- DATOS DE ENTRADA B ----------");
        testsEjercicio1("PI3E1B_DatosEntrada");
    }


    private static void testsEjercicio1(String fichero) {
        String rutaFichero = "ficheros/" + fichero + ".txt";
        String carpetaResultados = "resultados/ejercicio1/";

        Graph<Persona,Relacion> g = GraphsReader.newGraph(rutaFichero,
                Persona::ofFormat, Relacion::ofFormat,
                Graphs2::simpleDirectedGraph);
```

```
        // Grafo original
        GraphColors.toDot(g,
                carpetaResultados + fichero + ".gv",
                p -> p.toString(),
                r -> "",
                p -> GraphColors.color(Color.black),
                r -> GraphColors.color(Color.black));

        System.out.println("- Fichero procesado en " + fichero + ".gv en la carpeta " + carpetaResultados);

        System.out.println("\nAPARTADO A:");
        System.out.println("- Personas cuyos nombres aparecen en el grafo y cumplen los requisitos: " +
                Ejercicio1.fEjercicio1A(g, fichero).stream().map(x -> x.nombre()).toList());

        System.out.println("\nAPARTADO B:");
        if(fichero.equals("PI3E1A_DatosEntrada")) {
            Persona p = Persona.of(13, "Maria", 2008, "Sevilla");
            System.out.println("- Ancestros de " + p.nombre() + ": " +
                    Ejercicio1.fEjercicio1B(g, p, fichero).stream().map(x -> x.nombre()).toList());
        } else {
            Persona p = Persona.of(13, "Raquel", 1993, "Sevilla");
            System.out.println("- Ancestros de " + p.nombre() + ": " +
                    Ejercicio1.fEjercicio1B(g, p, fichero).stream().map(x -> x.nombre()).toList());
        }

        System.out.println("\nAPARTADO C:");
        if(fichero.equals("PI3E1A_DatosEntrada")) {
            Persona p1 = Persona.of(16, "Rafael", 2020, "Malaga");
            Persona p2 = Persona.of(14, "Sara", 2015, "Jaen");
            Persona p3 = Persona.of(13, "Maria", 2008, "Sevilla");
            Persona p4 = Persona.of(12, "Patricia", 2010, "Cordoba");
            Persona p5 = Persona.of(8, "Carmen", 1989, "Jaen");
            System.out.println("- " + p1.nombre() + " y " + p2.nombre() + " son " + Ejercicio1.fEjercicio1C(g, p1, p2));
            System.out.println("- " + p3.nombre() + " y " + p4.nombre() + " son " + Ejercicio1.fEjercicio1C(g, p3, p4));
            System.out.println("- " + p5.nombre() + " y " + p1.nombre() + " son " + Ejercicio1.fEjercicio1C(g, p5, p1));
        } else {
            Persona p1 = Persona.of(14, "Julia", 1996, "Jaen");
            Persona p2 = Persona.of(6, "Angela", 1997, "Sevilla");
            Persona p3 = Persona.of(15, "Alvaro", 2000, "Sevilla");
            Persona p4 = Persona.of(13, "Raquel", 1993, "Sevilla");
            Persona p5 = Persona.of(3, "Laura", 1965, "Jerez");
            System.out.println("- " + p1.nombre() + " y " + p2.nombre() + " son " + Ejercicio1.fEjercicio1C(g, p1, p2));
            System.out.println("- " + p3.nombre() + " y " + p4.nombre() + " son " + Ejercicio1.fEjercicio1C(g, p3, p4));
            System.out.println("- " + p5.nombre() + " y " + p4.nombre() + " son " + Ejercicio1.fEjercicio1C(g, p5, p4));
        }

        System.out.println("\nAPARTADO D:");
        System.out.println("- Personas que tienen hijos/as con distintas personas: " +
                Ejercicio1.fEjercicio1D(g, fichero).stream().map(x -> x.nombre()).toList());

        System.out.println("\nAPARTADO E:");
        System.out.println("- Personas de tal manera que se cubren todas la relaciones existentes: " +
                Ejercicio1.fEjercicio1E(g, fichero).stream().map(x -> x.nombre()).toList());
    }

}
```

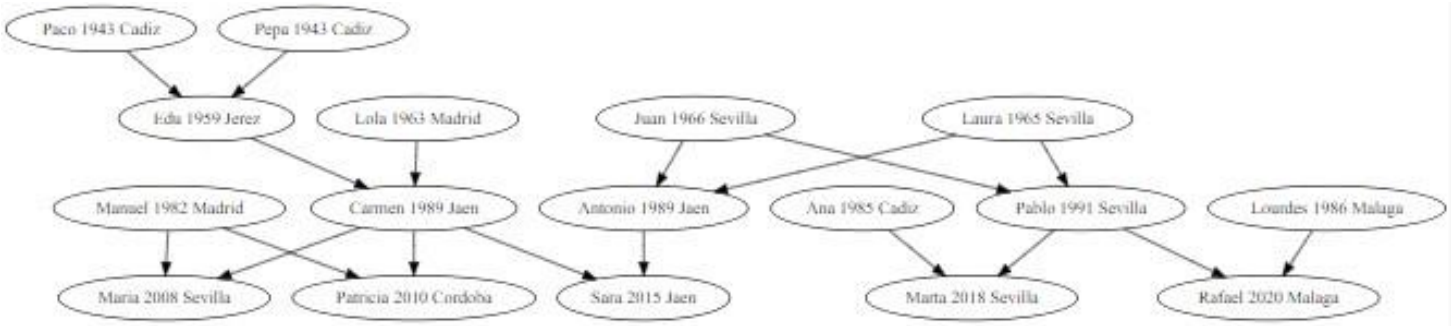
1.3. VOLCADO DE PANTALLA

```
* TEST EJERCICIO 1 *

---------- DATOS DE ENTRADA A ----------
- Fichero procesado en PI3E1A_DatosEntrada.gv en la carpeta resultados/ejercicio1/

APARTADO A:
- Fichero procesado en PI3E1A_DatosEntrada_ApartadoA.gv en la carpeta resultados/ejercicio1/
- Personas cuyos nombres aparecen en el grafo y cumplen los requisitos: [Edu, Sara]

APARTADO B:
- Fichero procesado en PI3E1A_DatosEntrada_ApartadoB.gv en la carpeta resultados/ejercicio1/
- Ancestros de Maria: [Manuel, Pepa, Paco, Edu, Lola, Carmen]

APARTADO C:
- Rafael y Sara son PRIMOS
- Maria y Patricia son HERMANOS
- Carmen y Rafael son OTROS

APARTADO D:
- Fichero procesado en PI3E1A_DatosEntrada_ApartadoD.gv en la carpeta resultados/ejercicio1/
- Personas que tienen hijos/as con distintas personas: [Pablo, Carmen]

APARTADO E:
- Fichero procesado en PI3E1A_DatosEntrada_ApartadoE.gv en la carpeta resultados/ejercicio1/
- Personas de tal manera que se cubren todas la relaciones existentes: [Carmen, Pablo, Antonio, Edu, Manuel, Ana, Rafael]
```

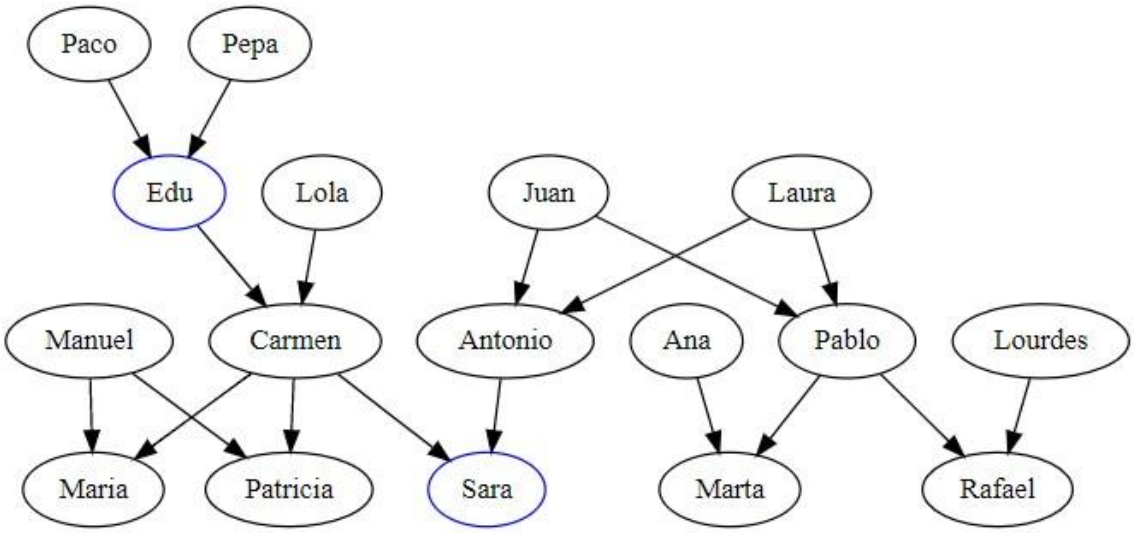
```
---------- DATOS DE ENTRADA B ----------
- Fichero procesado en PI3E1B_DatosEntrada.gv en la carpeta resultados/ejercicio1/

APARTADO A:
- Fichero procesado en PI3E1B_DatosEntrada_ApartadoA.gv en la carpeta resultados/ejercicio1/
- Personas cuyos nombres aparecen en el grafo y cumplen los requisitos: [Raquel, Angela, Josefina, Julia]

APARTADO B:
- Fichero procesado en PI3E1B_DatosEntrada_ApartadoB.gv en la carpeta resultados/ejercicio1/
- Ancestros de Raquel: [Irene, Daniel, Encarna, Pedro, Manuela, Ramon, Francisco, Josefina]

APARTADO C:
- Julia y Angela son PRIMOS
- Alvaro y Raquel son HERMANOS
- Laura y Raquel son OTROS

APARTADO D:
- Fichero procesado en PI3E1B_DatosEntrada_ApartadoD.gv en la carpeta resultados/ejercicio1/
- Personas que tienen hijos/as con distintas personas: [Josefina]

APARTADO E:
- Fichero procesado en PI3E1B_DatosEntrada_ApartadoE.gv en la carpeta resultados/ejercicio1/
- Personas de tal manera que se cubren todas la relaciones existentes: [Josefina, Ramon, Laura, Pedro, Marcos, Javier]
```

1.4. GRAFOS

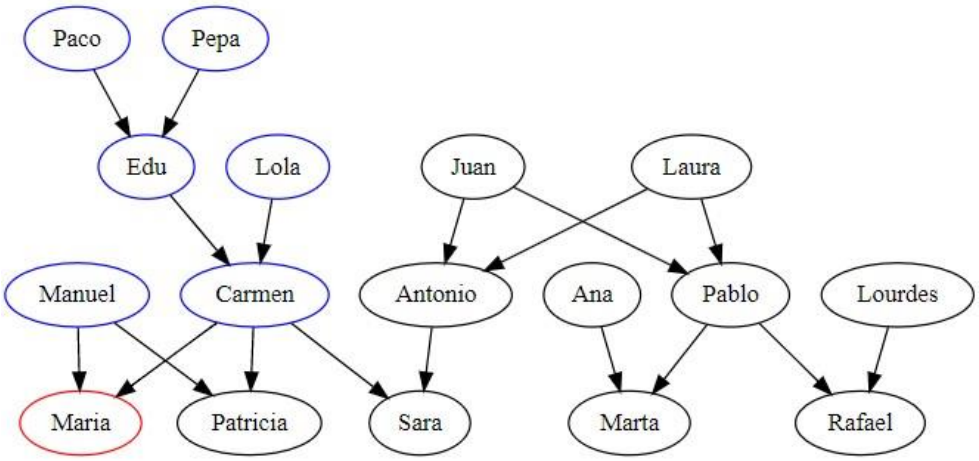
DATOS DE ENTRADA A:



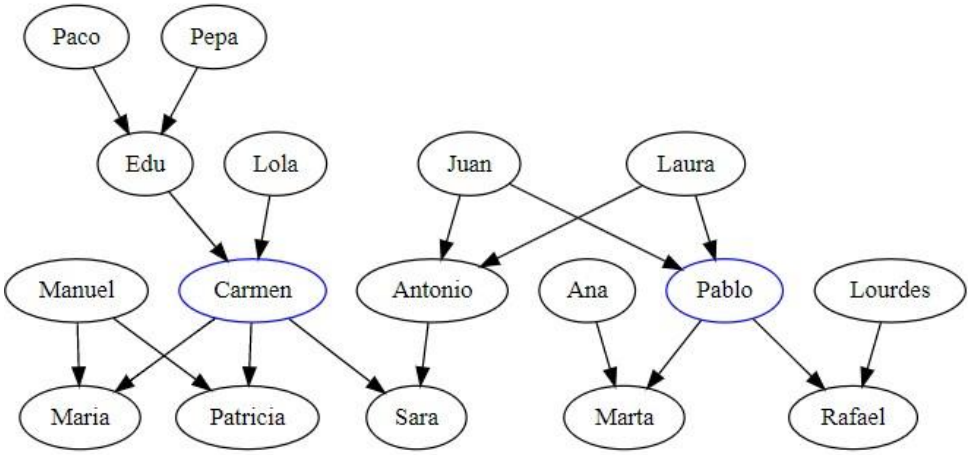
- APARTADO A



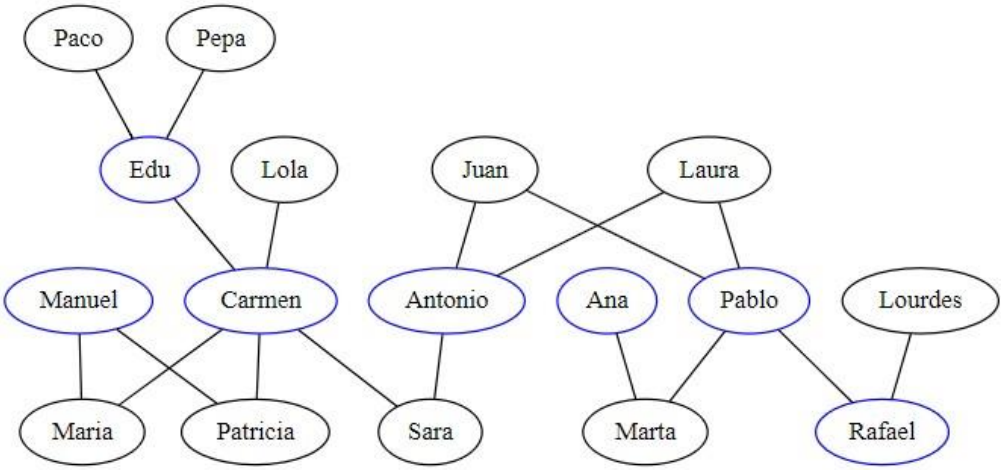
- APARTADO B



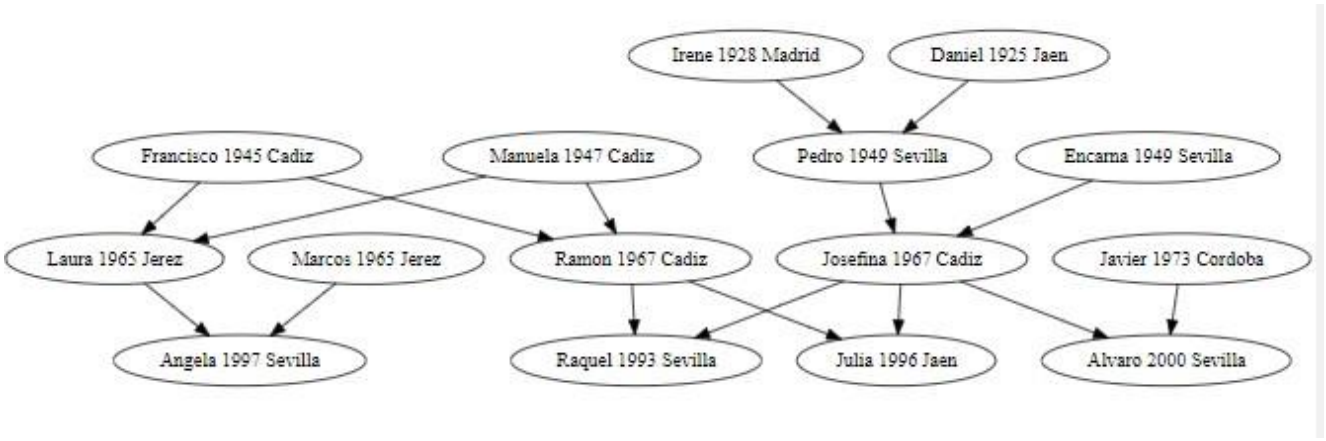
- APARTADO D



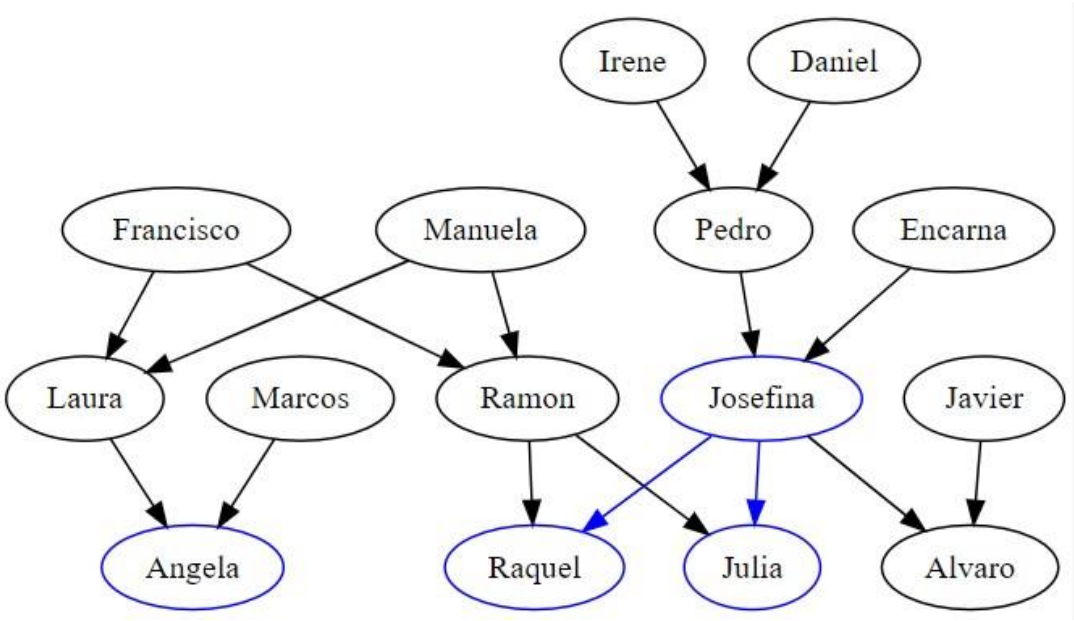
- APARTADO E



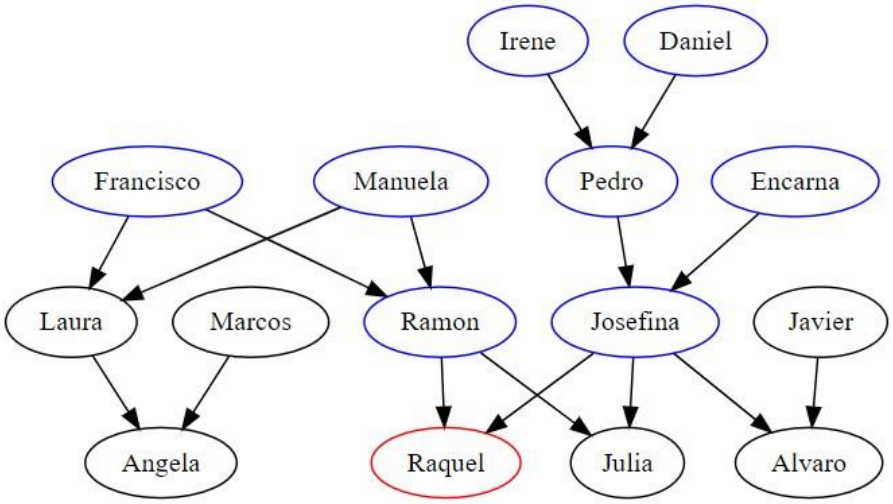
DATOS DE ENTRADA B:



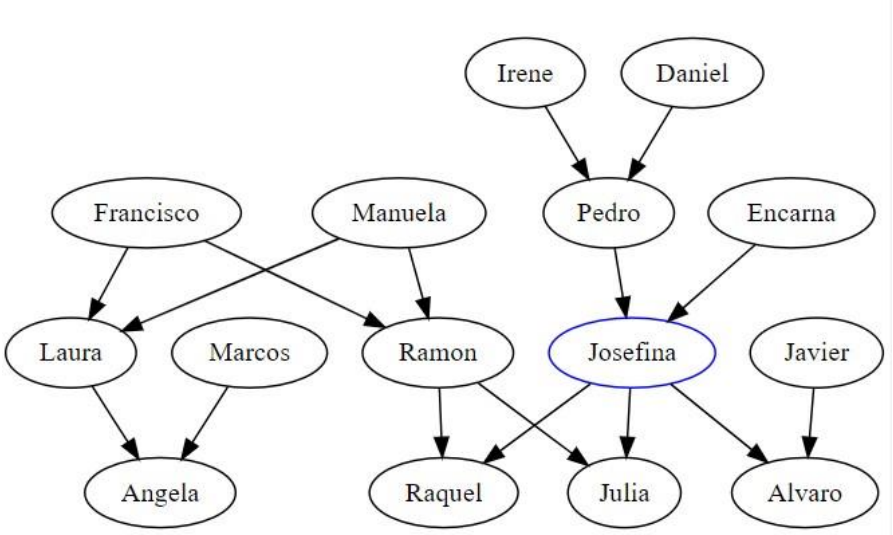
- APARTADO A



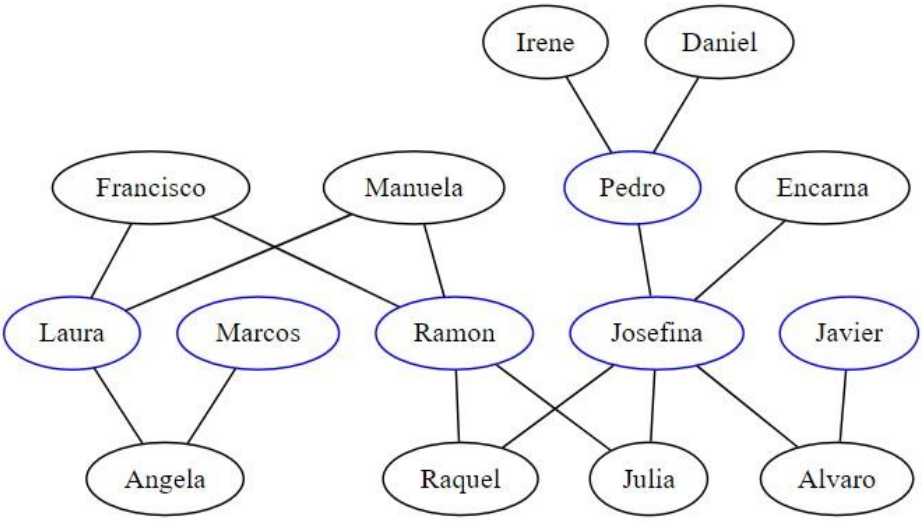
- APARTADO B



- APARTADO D



- APARTADO E



2. EJERCICIO 2

2.1. CÓDIGO EJERCICIO

```java
package ejercicios;

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

import org.jgrapht.Graph;
import org.jgrapht.GraphPath;
import org.jgrapht.alg.connectivity.ConnectivityInspector;
import org.jgrapht.alg.shortestpath.DijkstraShortestPath;
import org.jgrapht.alg.tour.HeldKarpTSP;

import tipos.Ciudad;
import tipos.Trayecto;
import us.lsi.colors.GraphColors;
import us.lsi.colors.GraphColors.Color;
import us.lsi.common.Pair;
import us.lsi.common.Trio;
import us.lsi.graphs.views.SubGraphView;

public class Ejercicio2 {

    private static String carpetaResultados = "resultados/ejercicio2/";

    // APARTADO A
    public static List<Set<Ciudad>> fEjercicio2A(Graph<Ciudad,Trayecto> gf, String fichero) {
        var alg = new ConnectivityInspector<>(gf);
        List<Set<Ciudad>> res = alg.connectedSets();

        GraphColors.toDot(gf,
                carpetaResultados + fichero + "_ApartadoA.gv",
                c -> c.nombre(),
                t -> "",
                c -> GraphColors.color(coloreadoCiudad(c, res)),
                t -> GraphColors.color(coloreadoTrayecto(gf, t, res)));

        System.out.println("- Fichero procesado en " + fichero + "_ApartadoA.gv en la carpeta " + carpetaResultados);
        return res;
    }

    private static Color coloreadoCiudad(Ciudad ciudad, List<Set<Ciudad>> ls) {
        Color color = null;
        for(Set<Ciudad> sc: ls) {
            if(sc.contains(ciudad)) {
                    color = Color.values()[ls.indexOf(sc)];
            }
        }
        return color;
    }

    private static Color coloreadoTrayecto(Graph<Ciudad, Trayecto> gf, Trayecto trayecto,
            List<Set<Ciudad>> ls) {
        Ciudad ciudad = gf.getEdgeSource(trayecto);
        return coloreadoCiudad(ciudad, ls);
    }


    // APARTADO B
    public static Set<Ciudad> fEjercicio2B(Graph<Ciudad,Trayecto> gf, String fichero) {
        List<Set<Ciudad>> datos = fEjercicio2A(gf, fichero);

        // Valores de referencia
        Set<Ciudad> res = datos.get(0);
        Integer suma = res.stream().mapToInt(x -> x.puntuacion()).sum();

        // Actualizacion de valores
        for(Set<Ciudad> sc: datos) {
            Integer sumaPosible = sc.stream().mapToInt(x -> x.puntuacion()).sum();
            if(sumaPosible > suma) {
                res = sc;
                suma = sumaPosible;
            }
        }

        representaGrafo(gf, fichero, res);

        System.out.println("- Fichero procesado en " + fichero + "_ApartadoB.gv en la carpeta " + carpetaResultados);
        return res;
    }

    private static void representaGrafo(Graph<Ciudad, Trayecto> gf, String fichero, Set<Ciudad> ls) {
        GraphColors.toDot(gf,
                carpetaResultados + fichero + "_ApartadoB.gv",
                c -> c.toString(),
                t -> "",
                c -> GraphColors.colorIf(Color.blue, ls.contains(c)),
                t -> GraphColors.colorIf(Color.blue, ls.contains(gf.getEdgeSource(t))));
    }
```

```java
// APARTADO C
public static Pair<List<Ciudad>, Double> fEjercicio2C(Graph<Ciudad,Trayecto> gf, String fichero) {
    var alg = new HeldKarpTSP<Ciudad, Trayecto>();
    List<Set<Ciudad>> datos = fEjercicio2A(gf, fichero);

    // Valores referencia
    Set<Ciudad> ciudades = datos.get(0);
    Graph<Ciudad,Trayecto> sfg = SubGraphView.of(gf, ciudades);
    GraphPath<Ciudad,Trayecto> tour = alg.getTour(sfg);
    Double precio = tour.getWeight();

    // Actualizacion de valores
    for(Set<Ciudad> sc: datos) {
        Graph<Ciudad,Trayecto> sfgPosible = SubGraphView.of(gf, sc);
        GraphPath<Ciudad,Trayecto> tourPosible = alg.getTour(sfgPosible);
        Double precioPosible = tourPosible.getWeight();
        if(precioPosible < precio) {
            ciudades = sc;
            tour = tourPosible;
            precio = precioPosible;
        }
    }

    representaGrafo2(gf, fichero, tour);

    System.out.println("- Fichero procesado en " + fichero + "_ApartadoC.gv en la carpeta " + carpetaResultados);
    return Pair.of(tour.getVertexList(), precio);
}

private static void representaGrafo2(Graph<Ciudad, Trayecto> gf, String fichero, GraphPath<Ciudad, Trayecto> tour) {
    GraphColors.toDot(gf,
            carpetaResultados + fichero + "_ApartadoC.gv",
            c -> c.nombre(),
            t -> t.precio().toString() + " euros",
            c -> GraphColors.colorIf(Color.blue, tour.getVertexList().contains(c)),
            t -> GraphColors.colorIf(Color.blue, tour.getEdgeList().contains(t)));
}

// APARTADO D
public static Trio<Ciudad, Ciudad, Double> fEjercicio2D(Graph<Ciudad, Trayecto> gf, String fichero,
        Set<Ciudad> ciudades, Integer num) {
    var alg = new DijkstraShortestPath<>(gf);
    List<Ciudad> ciudadesLista = ciudades.stream().toList();

    // Valores referencia
    GraphPath<Ciudad, Trayecto> camino = null;
    Double tiempo = Double.MAX_VALUE;

    // Actualizacion de valores
    for(int i=0; i<ciudades.size(); i++) {
        for(int j=i+1; j<ciudades.size(); j++) {
            if(!gf.containsEdge(ciudadesLista.get(i), ciudadesLista.get(j))) {
                GraphPath<Ciudad, Trayecto> caminoPosible = alg.getPath(ciudadesLista.get(i), ciudadesLista.get(j));
                Double tiempoPosible = caminoPosible.getWeight();
                if(tiempo > tiempoPosible) {
                    camino = caminoPosible;
                    tiempo = tiempoPosible;
                }
            }
        }
    }

    representaGrafo3(gf, fichero, camino, num);

    System.out.println("- Fichero procesado en " + fichero + "_ApartadoD.gv en la carpeta " + carpetaResultados);
    return Trio.of(camino.getStartVertex(), camino.getEndVertex(), tiempo);
}

private static void representaGrafo3(Graph<Ciudad, Trayecto> gf, String fichero,
        GraphPath<Ciudad, Trayecto> camino, Integer i) {
    GraphColors.toDot(gf,
            carpetaResultados + fichero + "_ApartadoD" + i + ".gv",
            c -> c.nombre(),
            t -> t.duracion().toString() + "minutos",
            c -> GraphColors.colorIf(Color.red, camino.getVertexList().contains(c)),
            t -> GraphColors.colorIf(Color.red, camino.getEdgeList().contains(t)));
}
```

```java
package tipos;


public record Ciudad(String nombre, Integer puntuacion) {

    public static Ciudad of(String nombre, Integer puntuacion) {
        return new Ciudad(nombre, puntuacion);
    }


    public static Ciudad ofFormat(String[] partes) {
        String nombre = partes[0].trim();
        Integer puntuacion = parseaPuntuacion(partes[1].trim());
        return Ciudad.of(nombre, puntuacion);
    }

    private static Integer parseaPuntuacion(String cadena) {
        String res = cadena.substring(0, cadena.indexOf("p"));
        return Integer.valueOf(res);
    }


    @Override
    public String toString() {
        return this.nombre + " " + this.puntuacion + " puntos";
    }


}
```

```java
package tipos;

public record Trayecto(Integer id, Double precio, Double duracion) {

    private static Integer numId = 0;

    public static Trayecto of(Double precio, Double duracion) {
        Integer id = numId;
        numId += 1;
        return new Trayecto(id, precio, duracion);
    }


    public static Trayecto ofFormat(String[] partes) {
        Double precio = parseaPrecio(partes[2].trim());
        Double duracion = parseaDuracion(partes[3].trim());
        return Trayecto.of(precio, duracion);
    }

    private static Double parseaPrecio(String cadena) {
        String res = cadena.substring(0, cadena.indexOf("e"));
        return Double.valueOf(res);
    }

    private static Double parseaDuracion(String cadena) {
        String res = cadena.substring(0, cadena.indexOf("m"));
        return Double.valueOf(res);
    }


    @Override
    public String toString() {
        return this.precio + " euros" + " " + this.duracion + " minutos";
    }


}
```

2.2. CÓDIGO TEST

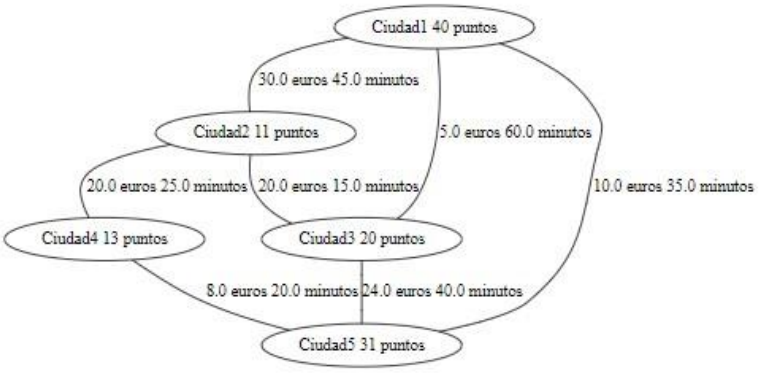
```
package tests;

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

import org.jgrapht.Graph;

import ejercicios.Ejercicio2;
import tipos.Ciudad;
import tipos.Trayecto;
import us.lsi.colors.GraphColors;
import us.lsi.colors.GraphColors.Color;
import us.lsi.common.Pair;
import us.lsi.common.Trio;
import us.lsi.graphs.Graphs2;
import us.lsi.graphs.GraphsReader;

public class TestEjercicio2 {

    public static void main(String[] args) {
        System.out.println("* TEST EJERCICIO 2 *");
        testsEjercicio2("PI3E2_DatosEntrada");
    }


    private static void testsEjercicio2(String fichero) {
        String rutaFichero = "ficheros/" + fichero + ".txt";
        String carpetaResultados = "resultados/ejercicio2/";

        Graph<Ciudad,Trayecto> g1 = GraphsReader.newGraph(rutaFichero,
                Ciudad::ofFormat, Trayecto::ofFormat,
                Graphs2::simpleWeightedGraph,
                Trayecto::precio);

        Graph<Ciudad,Trayecto> g2 = GraphsReader.newGraph(rutaFichero,
                Ciudad::ofFormat, Trayecto::ofFormat,
                Graphs2::simpleWeightedGraph,
                Trayecto::duracion);

        // Grafo original
        GraphColors.toDot(g1,
                carpetaResultados + fichero + ".gv",
                c -> c.toString(),
                t -> t.toString(),
                c -> GraphColors.color(Color.black),
                t -> GraphColors.color(Color.black));

        System.out.println("- Fichero procesado en " + fichero + ".gv en la carpeta " + carpetaResultados);

        System.out.println("\nAPARTADO A:");
        List<Set<Ciudad>> solucion = Ejercicio2.fEjercicio2A(g1, fichero);
        System.out.println("- Hay " + solucion.size() + " grupos de ciudades");
        for(int i=0; i<solucion.size(); i++) {
            System.out.println("- Grupo numero " + (i+1) + ": " +
                    solucion.get(i).stream().map(x -> x.nombre()).toList());
        }

        System.out.println("\nAPARTADO B:");
        System.out.println("- Grupo de ciudades que maximiza la suma de puntuaciones: " +
                Ejercicio2.fEjercicio2B(g2, fichero).stream().map(x -> x.nombre()).toList());

        System.out.println("\nAPARTADO C:");
        Pair<List<Ciudad>, Double> solucionPair = Ejercicio2.fEjercicio2C(g1, fichero);
        System.out.println("- Grupo de ciudades a visitar que dan lugar que dan lugar al camino cerrado de menor precio:");
        System.out.println(solucionPair.first().stream().map(x -> x.nombre()).toList() + " --> " + solucionPair.second() + " euros");

        System.out.println("\nAPARTADO D:");
        List<Set<Ciudad>> datos = Ejercicio2.fEjercicio2A(g2, fichero);
        for(int i=0; i<datos.size(); i++) {
            Trio<Ciudad, Ciudad, Double> solucionD = Ejercicio2.fEjercicio2D(g2, fichero, datos.get(i), i+1);
            System.out.println("- Para el grupo de ciudades " + datos.get(i).stream().map(x -> x.nombre()).toList() +
                    ", las ciudades no conectadas directamente entre las que se puede viajar en menor tiempo son: ");
            System.out.println("Origen: " + solucionD.first().nombre() + " y Destino: " + solucionD.second().nombre() +
                    " --> " + "Tiempo: " + solucionD.third() + " minutos");
        }
    }

}
```

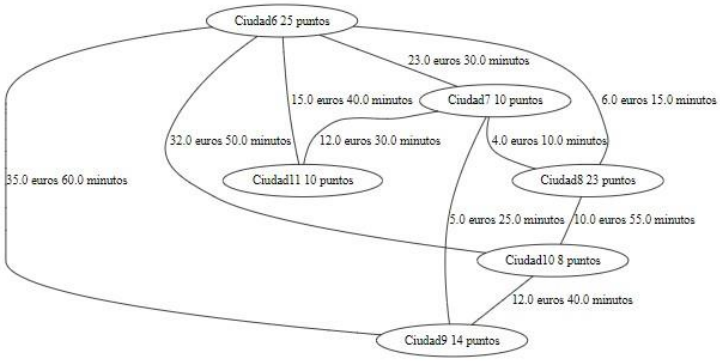
2.3. VOLCADO DE PANTALLA

```
* TEST EJERCICIO 2 *
- Fichero procesado en PI3E2_DatosEntrada.gv en la carpeta resultados/ejercicio2/

APARTADO A:
- Fichero procesado en PI3E2_DatosEntrada_ApartadoA.gv en la carpeta resultados/ejercicio2/
- Hay 2 grupos de ciudades
- Grupo numero 1: [Ciudad5, Ciudad2, Ciudad4, Ciudad3, Ciudad1]
- Grupo numero 2: [Ciudad8, Ciudad11, Ciudad10, Ciudad6, Ciudad7, Ciudad9]

APARTADO B:
- Fichero procesado en PI3E2_DatosEntrada_ApartadoA.gv en la carpeta resultados/ejercicio2/
- Fichero procesado en PI3E2_DatosEntrada_ApartadoB.gv en la carpeta resultados/ejercicio2/
- Grupo de ciudades que maximiza la suma de puntuaciones: [Ciudad5, Ciudad2, Ciudad4, Ciudad3, Ciudad1]

APARTADO C:
- Fichero procesado en PI3E2_DatosEntrada_ApartadoA.gv en la carpeta resultados/ejercicio2/
- Fichero procesado en PI3E2_DatosEntrada_ApartadoC.gv en la carpeta resultados/ejercicio2/
- Grupo de ciudades a visitar que dan lugar que dan lugar al camino cerrado de menor precio:
[Ciudad8, Ciudad10, Ciudad9, Ciudad7, Ciudad11, Ciudad6, Ciudad8] --> 60.0 euros

APARTADO D:
- Fichero procesado en PI3E2_DatosEntrada_ApartadoA.gv en la carpeta resultados/ejercicio2/
- Fichero procesado en PI3E2_DatosEntrada_ApartadoD.gv en la carpeta resultados/ejercicio2/
- Para el grupo de ciudades [Ciudad5, Ciudad2, Ciudad4, Ciudad3, Ciudad1], las ciudades no conectadas directamente entre las que se puede viajar en menor tiempo son:
Origen: Ciudad4 y Destino: Ciudad3 --> Tiempo: 40.0 minutos
- Fichero procesado en PI3E2_DatosEntrada_ApartadoD.gv en la carpeta resultados/ejercicio2/
- Para el grupo de ciudades [Ciudad8, Ciudad11, Ciudad10, Ciudad6, Ciudad7, Ciudad9], las ciudades no conectadas directamente entre las que se puede viajar en menor tiempo son:
Origen: Ciudad8 y Destino: Ciudad9 --> Tiempo: 35.0 minutos
```

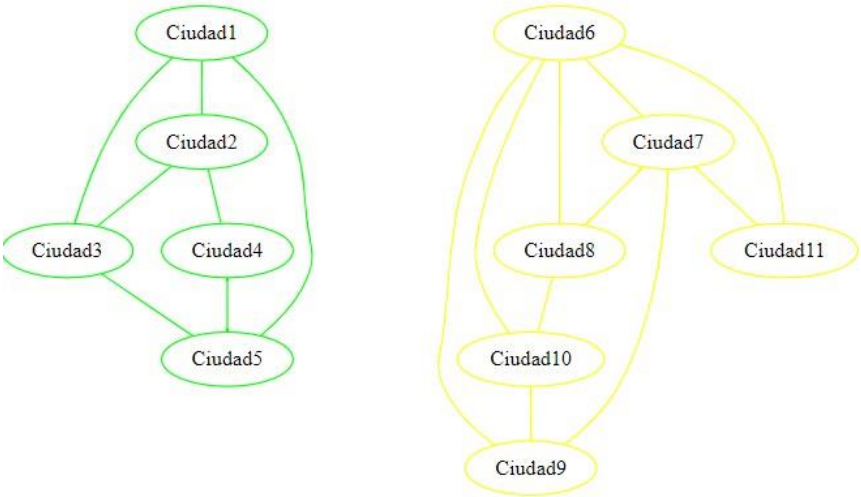
2.4. GRAFOS

- GRAFO ORIGINAL

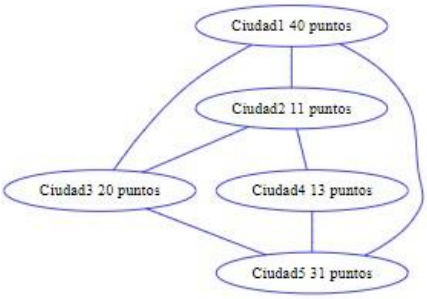


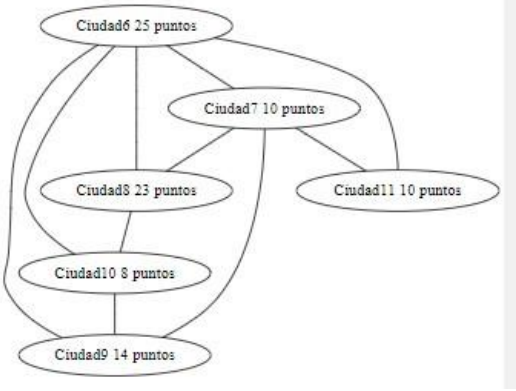


- APARTADO A

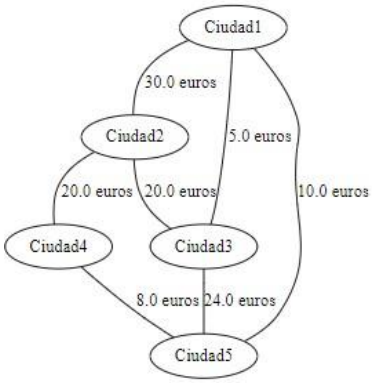


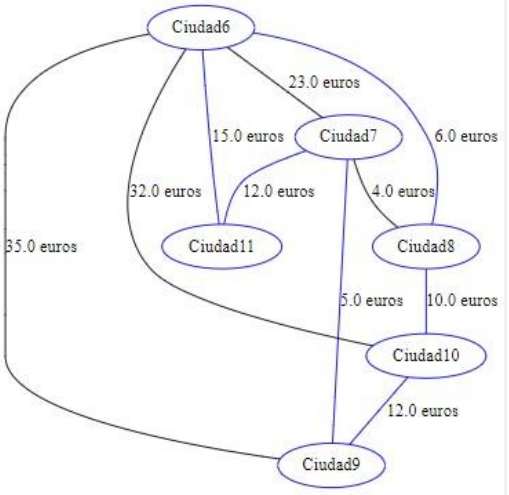
- APARTADO B



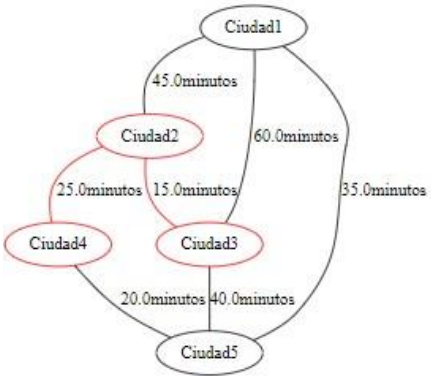


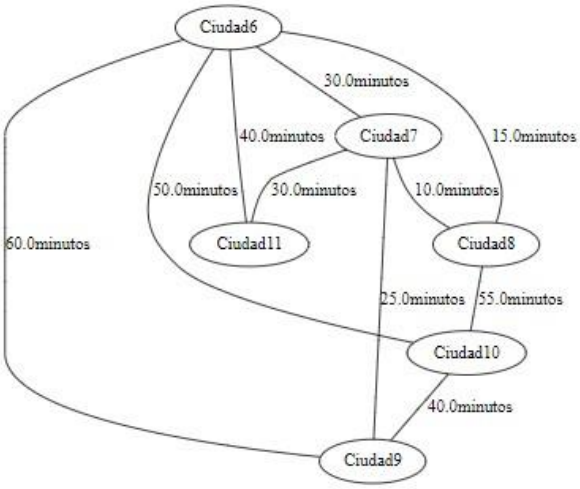
- APARTADO C

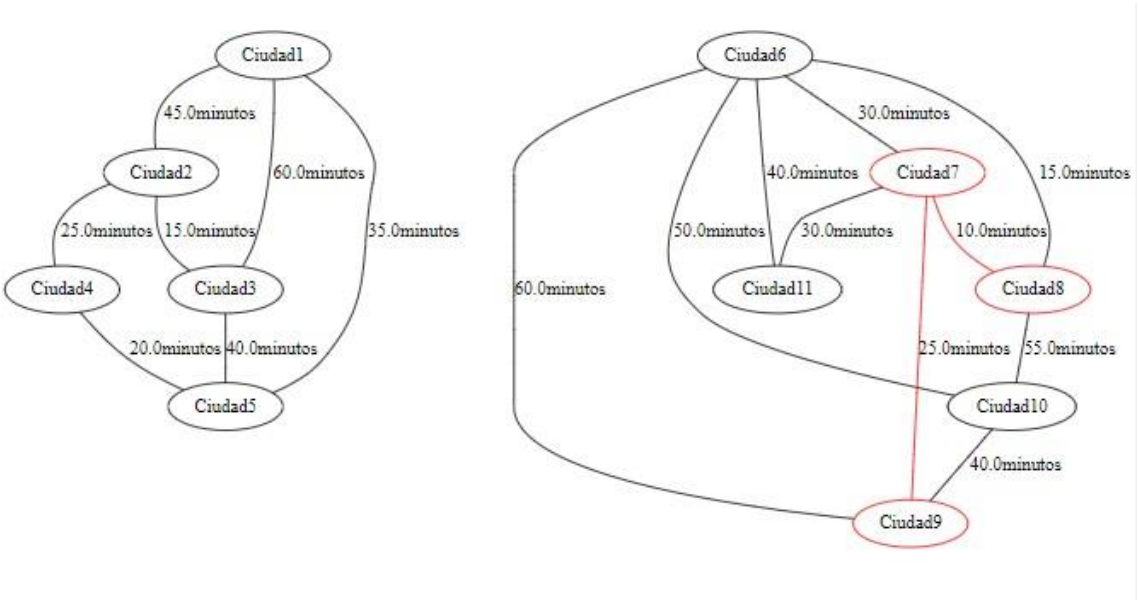




- APARTADO D







3. EJERCICIO 3

3.1. CÓDIGO EJERCICIO

```
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 tipos.Actividad;
import tipos.Conexion;
import us.lsi.colors.GraphColors;
import us.lsi.colors.GraphColors.Color;

public class Ejercicio3 {

    private static String carpetaResultados = "resultados/ejercicio3/";

    // APARTADO A
    public static List<Set<Actividad>> fEjercicio3A(Graph<Actividad,Conexion> gf, String fichero) {
        var alg = new GreedyColoring<>(gf);
        Coloring<Actividad> solucion = alg.getColoring();
        List<Set<Actividad>> res = solucion.getColorClasses();
        return res;
    }


   // APARTADO B
   public static void fEjercicio3B(Graph<Actividad,Conexion> gf, String fichero) {
       var alg = new GreedyColoring<>(gf);
       Coloring<Actividad> solucion = alg.getColoring();
       Map<Actividad,Integer> coloresPorActividad = solucion.getColors();

       GraphColors.toDot(gf,
               carpetaResultados + fichero + "_ApartadoB.gv",
               a -> a.nombre(),
               c -> "",
               a -> GraphColors.color(coloresPorActividad.get(a)),
               c -> GraphColors.color(Color.black));

       System.out.println("- Fichero procesado en " + fichero + "_ApartadoB.gv en la carpeta " + carpetaResultados);
   }

}
```

```java
package tipos;

public record Actividad(String nombre) {

    public static Actividad of(String nombre) {
        return new Actividad(nombre);
    }

    @Override
    public String toString() {
        return this.nombre;
    }

}
```

```java
package tipos;

public record Conexion(Integer id) {

    private static Integer numId = 0;

    public static Conexion of() {
        Integer id = numId;
        numId += 1;
        return new Conexion(id);
    }

    public static Conexion ofFormat(String[] partes) {
        return Conexion.of();
    }

}
```

3.2. CÓDIGO TEST

```java
package tests;

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

import org.jgrapht.Graph;

import ejercicios.Ejercicio3;
import tipos.Actividad;
import tipos.Conexion;
import us.lsi.colors.GraphColors;
import us.lsi.colors.GraphColors.Color;
import us.lsi.graphs.Graphs2;
import us.lsi.streams.Stream2;

public class TestEjercicio3 {

    public static void main(String[] args) {
        System.out.println("* TEST EJERCICIO 3 *");

        System.out.println("\n---------- DATOS DE ENTRADA A ----------");
        testsEjercicio3("PI3E3A_DatosEntrada");

        System.out.println("\n---------- DATOS DE ENTRADA B ----------");
        testsEjercicio3("PI3E3B_DatosEntrada");
    }


    private static void testsEjercicio3(String fichero) {
        String rutaFichero = "ficheros/" + fichero + ".txt";
        String carpetaResultados = "resultados/ejercicio3/";

        Graph<Actividad,Conexion> g = Graphs2.simpleGraph();

        fRellenaGrafo(g, rutaFichero);
```

```
        // Grafo original
        GraphColors.toDot(g,
                carpetaResultados + fichero + ".gv",
                a -> a.nombre(),
                c -> "",
                a -> GraphColors.color(Color.black),
                c -> GraphColors.color(Color.black));

        System.out.println("- Fichero procesado en " + fichero + ".gv en la carpeta " + carpetaResultados);

        System.out.println("\nAPARTADO A:");
        List<Set<Actividad>> sol = Ejercicio3.fEjercicio3A(g, fichero);
        System.out.println("- Numero de franjas horarias necesarias: " + sol.size());
        System.out.println("- Actividades para mostrarse en paralelo por franja horaria: ");
        for(int i=0; i<sol.size(); i++) {
            System.out.println("   Franja numero " + (i+1) + ": " + sol.get(i));
        }

        System.out.println("\nAPARTADO B:");
        Ejercicio3.fEjercicio3B(g, fichero);
    }
```

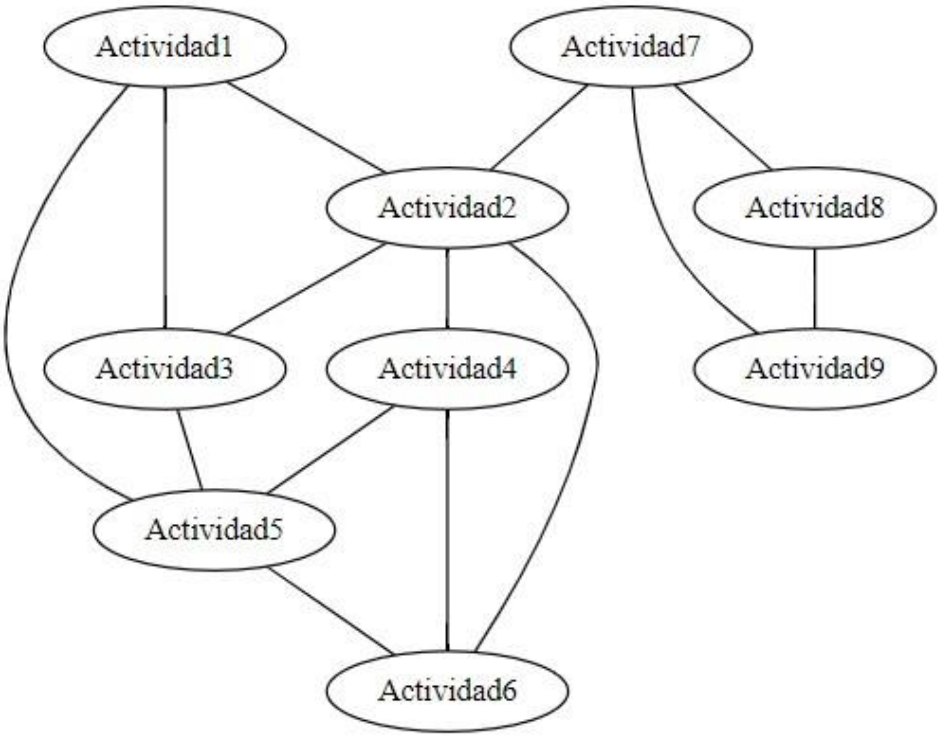
```
    private static void fRellenaGrafo(Graph<Actividad, Conexion> g, String rutaFichero) {
        List<String> datos = Stream2.file(rutaFichero).toList();

        for(String linea: datos) {
            String[] partes = linea.split(":");
            String datosPartes = partes[1].trim();
            String[] actividadesPartes = datosPartes.split(",");

            List<Actividad> actividades = new ArrayList<>();

            for(String actividadStr: actividadesPartes) {
                Actividad actividad = Actividad.of(actividadStr.trim());
                g.addVertex(actividad);
                actividades.add(actividad);
            }

            for(int i=0; i<actividades.size(); i++) {
                for(int j=i+1; j<actividades.size(); j++) {
                    g.addEdge(actividades.get(i), actividades.get(j), Conexion.of());
                }
            }
        }
    }

}
```

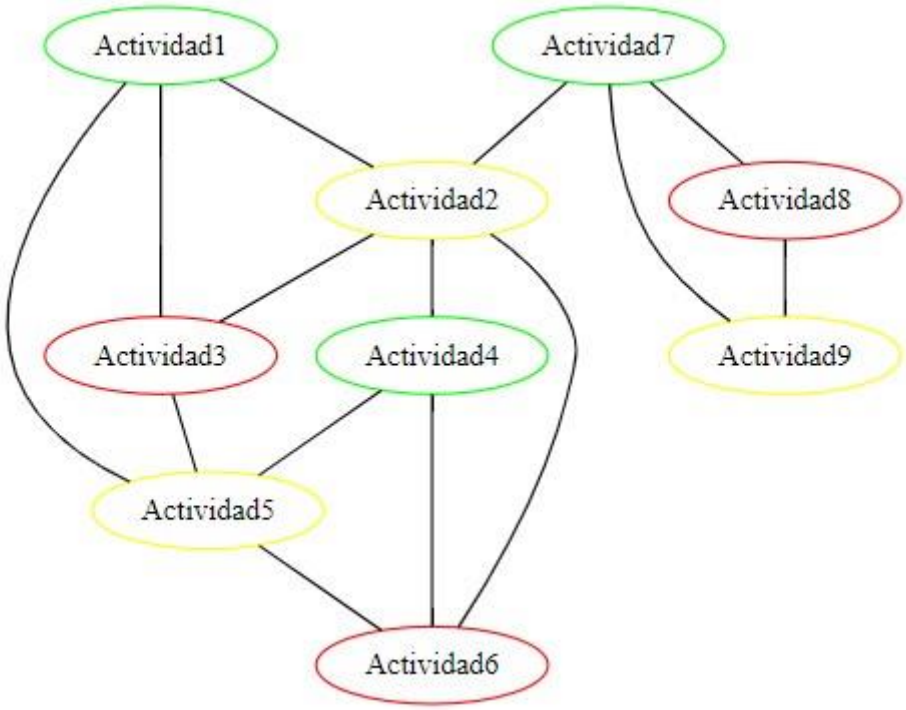
3.3. VOLCADO DE PANTALLA

```
* TEST EJERCICIO 3 *

---------- DATOS DE ENTRADA A ----------
- Fichero procesado en PI3E3A_DatosEntrada.gv en la carpeta resultados/ejercicio3/

APARTADO A:
- Numero de franjas horarias necesarias: 3
- Actividades para mostrarse en paralelo por franja horaria:
   Franja numero 1: [Actividad1, Actividad4, Actividad7]
   Franja numero 2: [Actividad2, Actividad9, Actividad5]
   Franja numero 3: [Actividad3, Actividad6, Actividad8]

APARTADO B:
- Fichero procesado en PI3E3A_DatosEntrada_ApartadoB.gv en la carpeta resultados/ejercicio3/

---------- DATOS DE ENTRADA B ----------
- Fichero procesado en PI3E3B_DatosEntrada.gv en la carpeta resultados/ejercicio3/

APARTADO A:
- Numero de franjas horarias necesarias: 4
- Actividades para mostrarse en paralelo por franja horaria:
   Franja numero 1: [Actividad1, Actividad11, Actividad8]
   Franja numero 2: [Actividad2, Actividad4, Actividad3, Actividad9, Actividad12, Actividad7]
   Franja numero 3: [Actividad10, Actividad5]
   Franja numero 4: [Actividad6]

APARTADO B:
- Fichero procesado en PI3E3B_DatosEntrada_ApartadoB.gv en la carpeta resultados/ejercicio3/
```

3.4. GRAFOS

DATOS DE ENTRADA A:





DATOS DE ENTRADA B: